

**CENG 491**

**INITIAL DESIGN REPORT**  
**HTML5 Canvas Workflow Diagram Editor**  
**iFlowEdit**

Sponsored by  
**INNOVA IT Solutions Inc.**

**TriUlti**

KARAOĞUZ, Mehmet Ozan

KAYRAK, Alaattin

KORKMAZ, Ozan

November 21, 2011

## Contents

1.	Introduction.....	4
1.1.	Problem Definition .....	4
1.2.	Purpose.....	4
1.3.	Scope .....	4
1.4.	Overview.....	5
1.5.	Definitions, Acronyms and Abbreviations .....	5
1.6.	References .....	6
2.	System Overview .....	6
3.	Design Consideration.....	7
3.1.	Design Assumptions, Dependencies and Constraints .....	7
3.2.	Design Goals and Guidelines .....	7
4.	Data Design.....	8
4.1.	Data Description .....	9
4.1.1.	User Information Data.....	9
4.1.2.	Server-Client Communication Data .....	9
4.1.3.	Document Information Data .....	10
4.1.4.	Configuration Data .....	10
4.2.	Data Dictionary .....	11
a.	ConnectionInstance.....	11
b.	ConnectionDefinition .....	13
c.	ActivityInstance .....	13
d.	ActivityDefinition.....	14
e.	WorkFlow .....	15
f.	FileController.....	15
g.	DatabaseController .....	16
5.	System Architecture .....	17
5.1.	Architectural Design .....	17
	.....	17
5.2.	Description of Components.....	18
5.2.1.	User .....	18
5.2.2.	Component Administrator .....	21
5.2.3.	Component Server.....	23
5.3.	Design Rationale .....	26

6.	User Interface Design .....	27
6.1.	Overview of User Interface .....	27
6.2.	Screen Images.....	27
6.2.1.	User Login Screen .....	27
6.2.2.	Document Selection Screen .....	27
6.2.3.	Design Area Screen.....	28
6.2.4.	Administrator Login Screen .....	29
6.2.5.	Administrator Area Screen .....	29
6.3.	Screen Objects and Actions .....	30
7.	Libraries and Tools.....	31
8.	Time Planning .....	32
8.1.	Term 1 Gantt Chart.....	32
8.2.	Term 2 Gantt Chart.....	32
9.	Conclusion .....	33

## **1. Introduction**

This document contains the software requirements for iFlowEdit which is an editor to create and edit work flow 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 work flow 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 work flow editors. On the other hand, in the market of work flow editors, there are many problems due to compatibility platforms and reaching all features. The iFlowEdit will be a work flow diagram editor which is platform independent and also does not require Adobe Flash. It only requires Internet browser regardless of operating system of device, which is why it gets rid of compatibility problems in the market. In addition, it is open source software, so anyone can easily reach its all features.

### **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 software system will be designed to maximize productivity of the editor by providing tools that assist in drawing work flow diagrams. More specifically, this system is designed to draw the

diagrams with templates and it keeps the projects in the database of the system. Therefore, user can load and reload the data to/from server to reach his workspace later.

## **1.4. Overview**

This document contains eight 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. In the seventh 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

- Projects @ Apache Documentation, <http://projects.apache.org/docs/index.html>
- Software Requirements Specification HTML5 Canvas Workflow Diagram Editor, TriUlti, Dec 25, 2011
- Oryx Configuration Specifications, <http://oryx-editor.googlecode.com/files/OryxSSS.pdf>
- JSON and XML, <http://www.json.org/xml.html>
- IEEE Std80, <http://standards.ieee.org/findstds/interps/80.html>
- Ext-JS usage, <http://docs.sencha.com/ext-js/4-0/>
- Processing.js references, <http://processingjs.org/reference>
- XAML explanations, <http://msdn.microsoft.com/en-us/library/ms752059.aspx>

## 2. System Overview

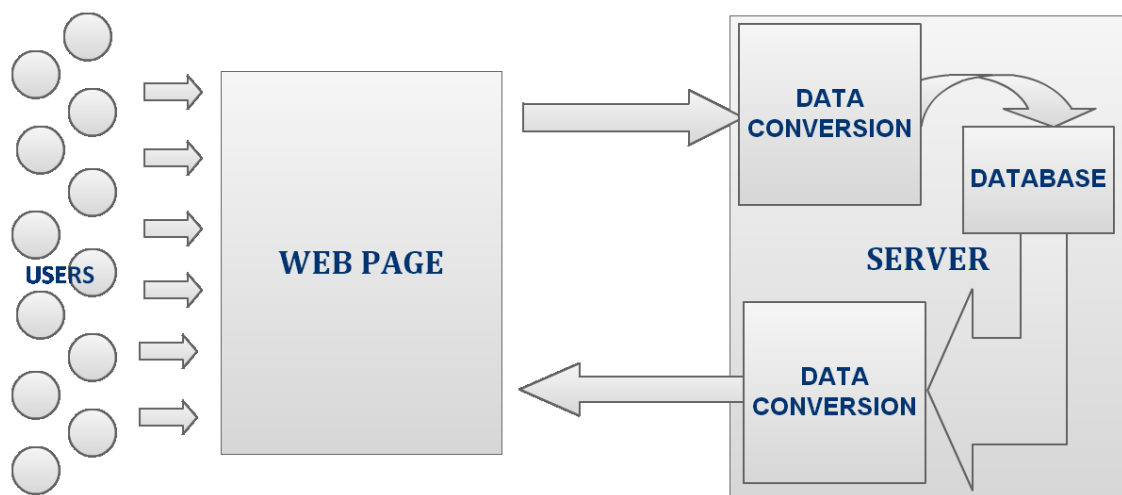


Figure 1 - System Overview

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 Consideration**

#### **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.

#### **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

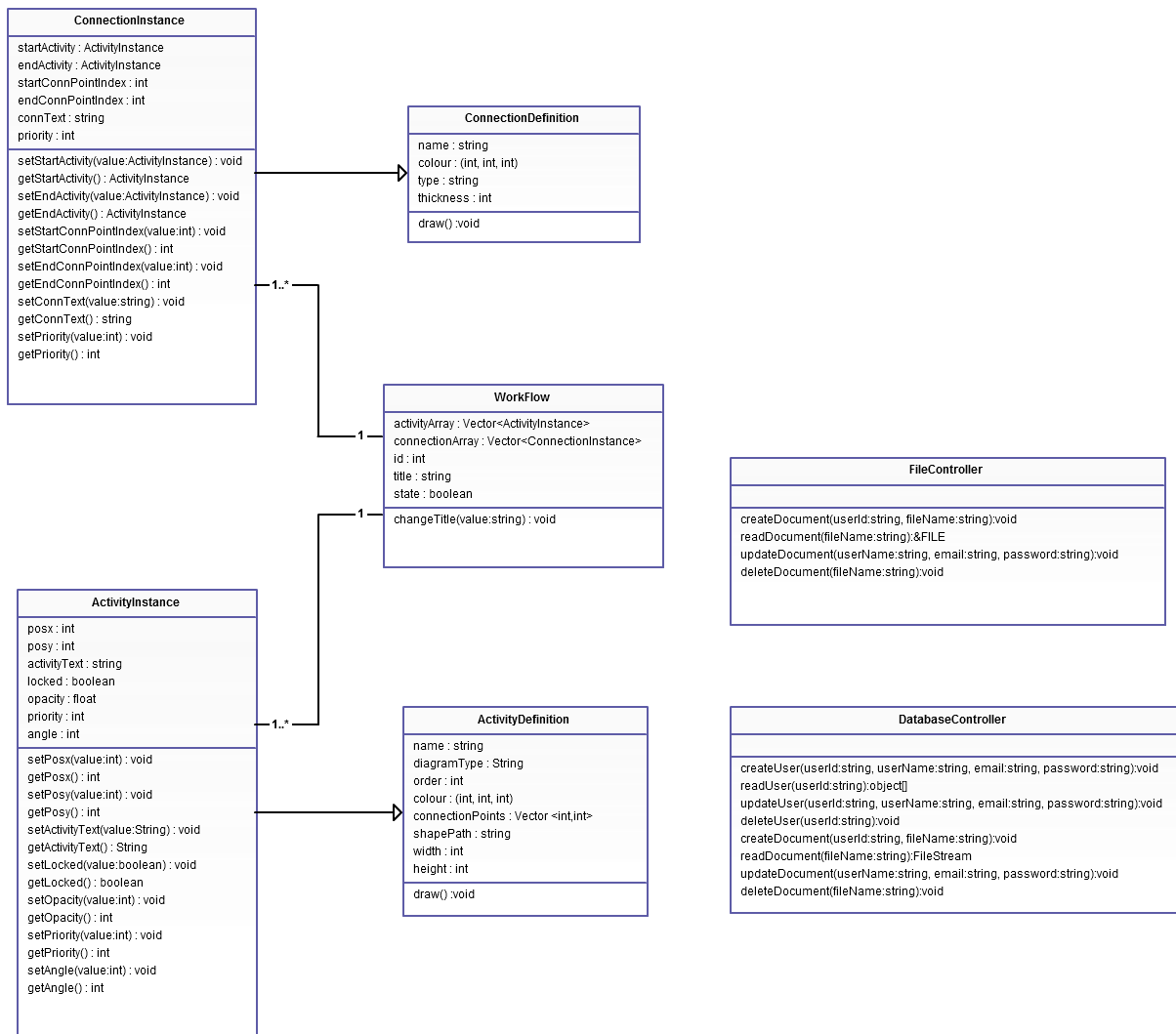


Figure 2 - Class Diagram of the System



## 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 3. 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.

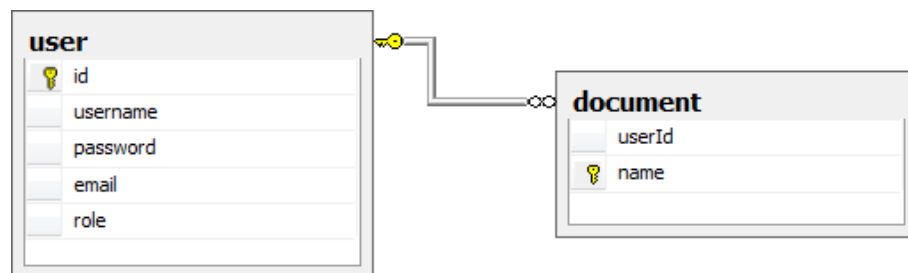


Figure 3 - Database Tables

### 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 4 shows the XML version and figure 5 shows the JSON version.

```

...
<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 4 - XML

```

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

```

Figure 5 – 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

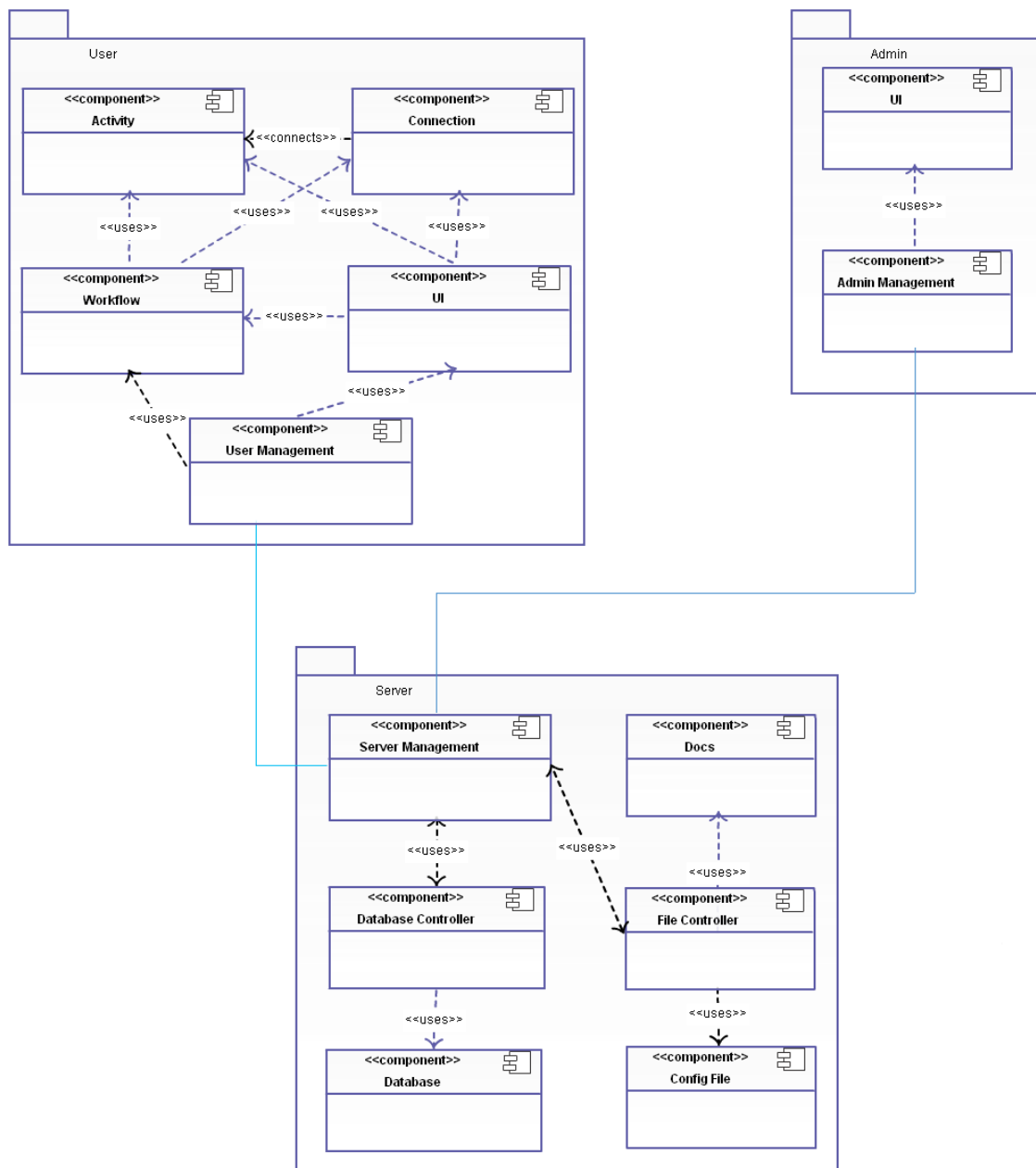


Figure 6 - Component Diagram

## 5.2. Description of Components

### 5.2.1. User

#### 5.2.1.1. *Processing Narrative for Administrator*

User component is responsible for performing every possible action that users of the iFlowEdit may do. These actions are creating work flows, editing work flows, saving work flows, loading work flows and deleting work flows. 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 work flow. 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 Work Flow sub-component on the account of the fact that every actions which are performed on the activities effects the work flow which contains that activities. Work flow sub-component applies the changes to work flow, then uses UI sub-component. Finally, UI sub-component gives the output to the user.

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 Work flow sub-component on the account of the fact that every actions which are performed on the connections effects the work flow which contains that connections. Work flow sub-component applies the changes to work flow, then uses UI sub-component. Finally, UI sub-component gives the output to the user.

Third one is that if a user gives an input directly about work flow, UI sub-component processed input, then uses Work Flow sub-component. Work flow sub-component applies the changes to work flow, then uses UI sub-component. Finally, UI sub-component gives the output to the user.

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.

#### 5.2.1.4. *Dynamic Behaviour of User*

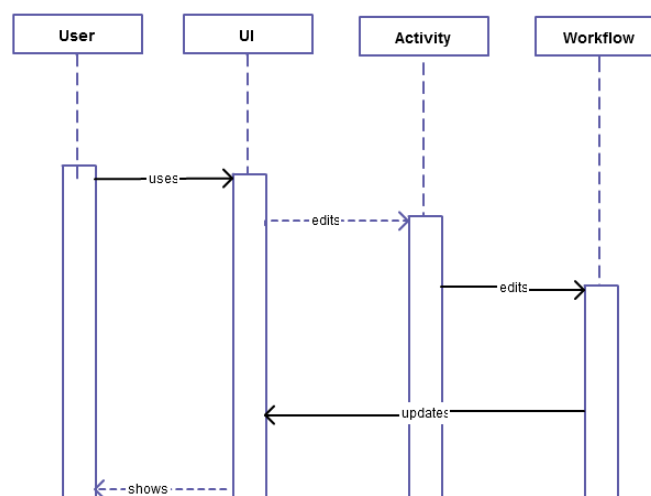


Figure 7 - Sequential Diagram of Use Case 1

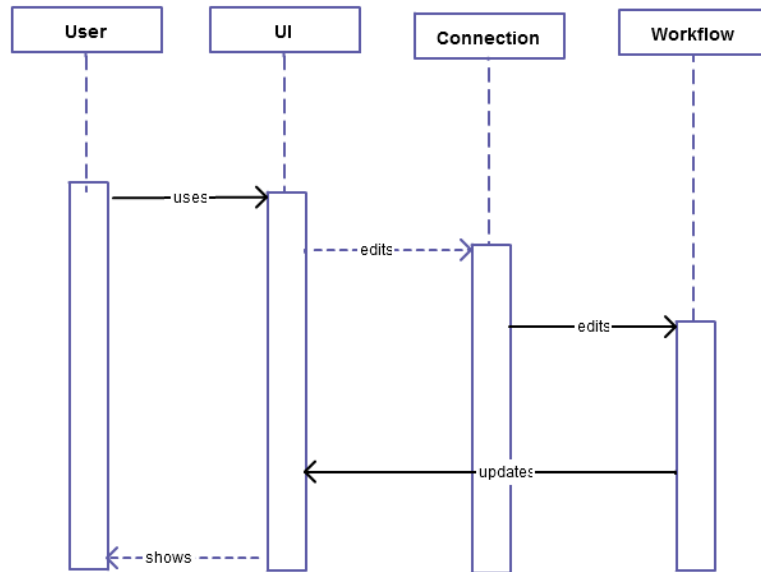


Figure 8 - Sequential Diagram of Use Case 2

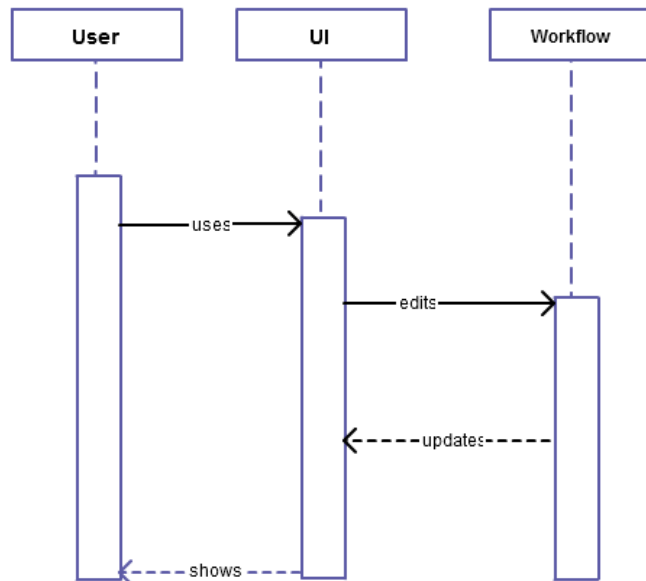


Figure 9 - Sequential Diagram of Use Case 3

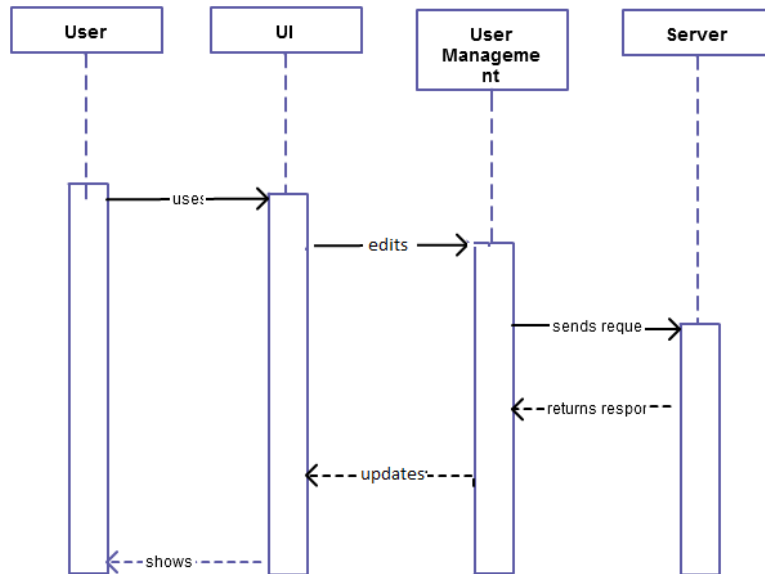


Figure 10 - Sequential Diagram of Use Case 4

## 5.2.2. Component Administrator

### 5.2.2.1. Processing Narrative for Administrator

Administrator component is responsible for performing every possible action that administrators 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 pieces of information which are user name, email and password on the database and an administrator can reach just two of them which are user name and email. However, on a create action, administrator should provide a password for the user.

### 5.2.2.2. Interface Description of User

Administrator component takes inputs from administrators of the system via Administrator GUI. The component which controls inputs that are taken from GUI is UI sub-component. For example, to create a user, an administrator 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 administrator.

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

#### 5.2.2.3. *Processing Detail of Administrator*

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

#### 5.2.2.4. *Dynamic Behaviour of Administrator*

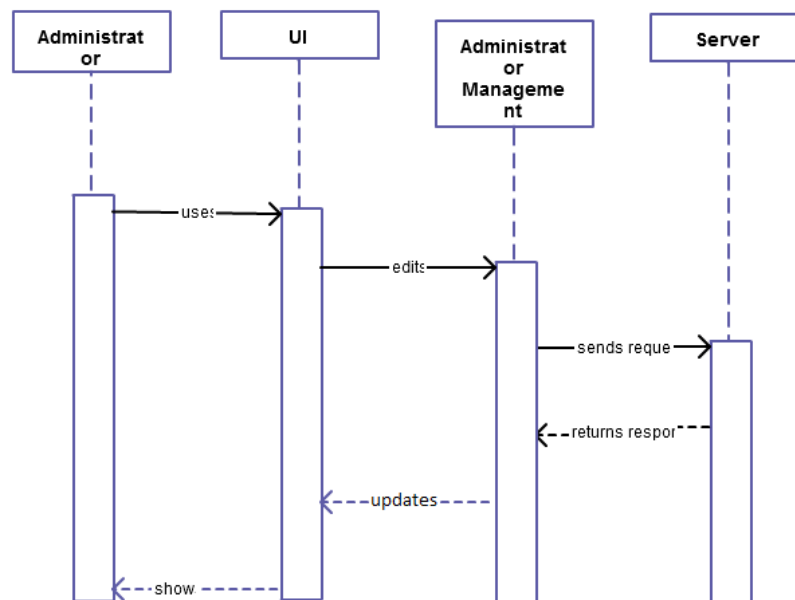


Figure 11 - Sequential Diagram of Administrator Use Case

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

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

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.

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.

Forth and the last one is that if an administrator sends any request, Administrator component interacts with Server Management sub-component. Server Management uses Database Controller to contact with Database. After that Server Management sends respond to Administrator Component.

#### 5.2.3.4. *Dynamic Behaviour of Server*

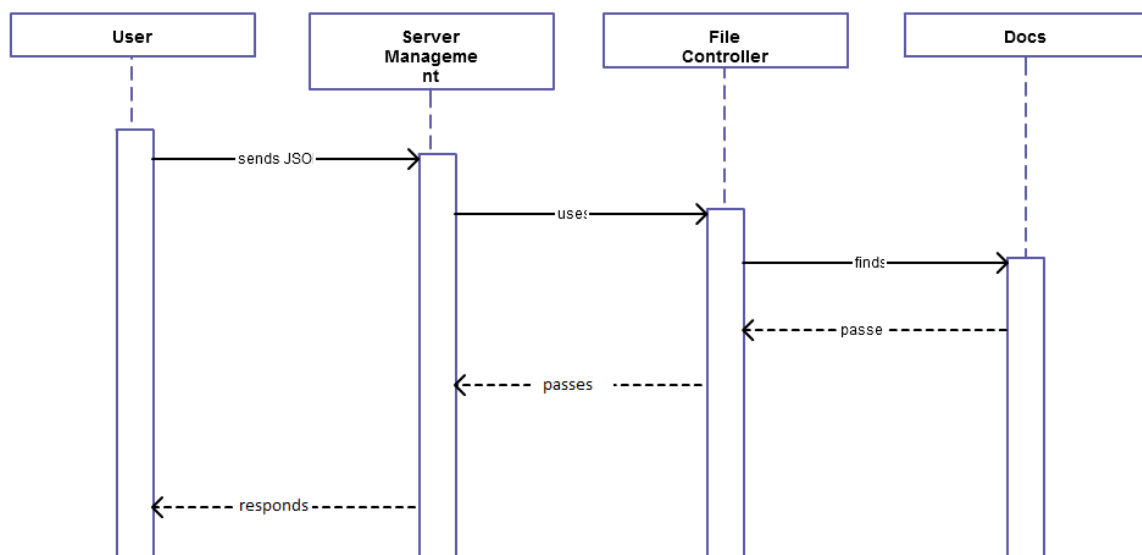


Figure 12 - Sequential Diagram of Server Use Case 1



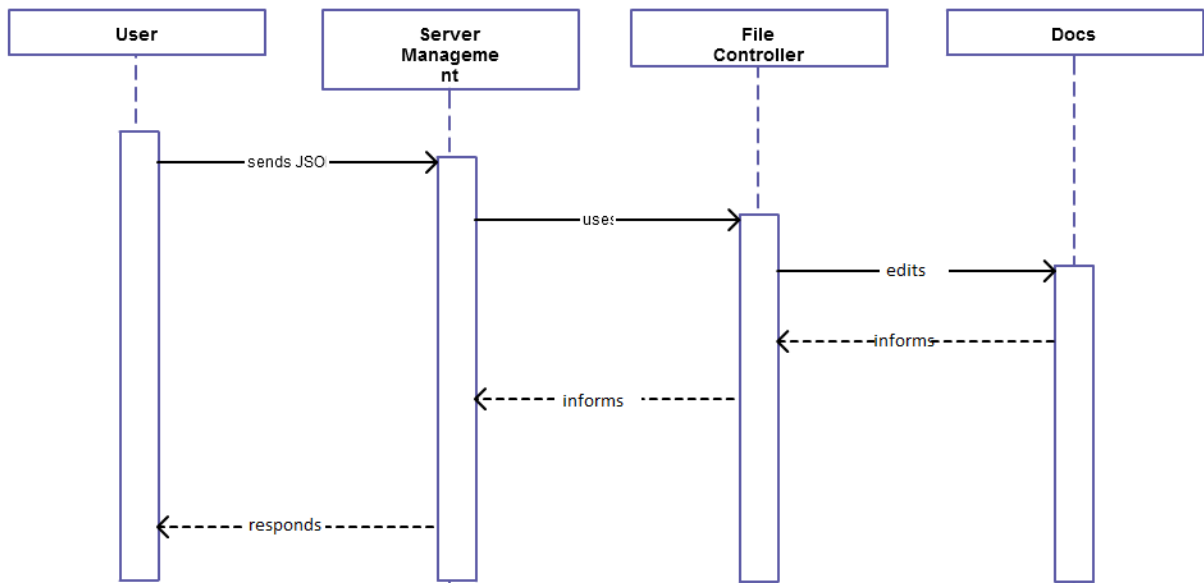


Figure 13 - Sequential Diagram of Server Use Case 2

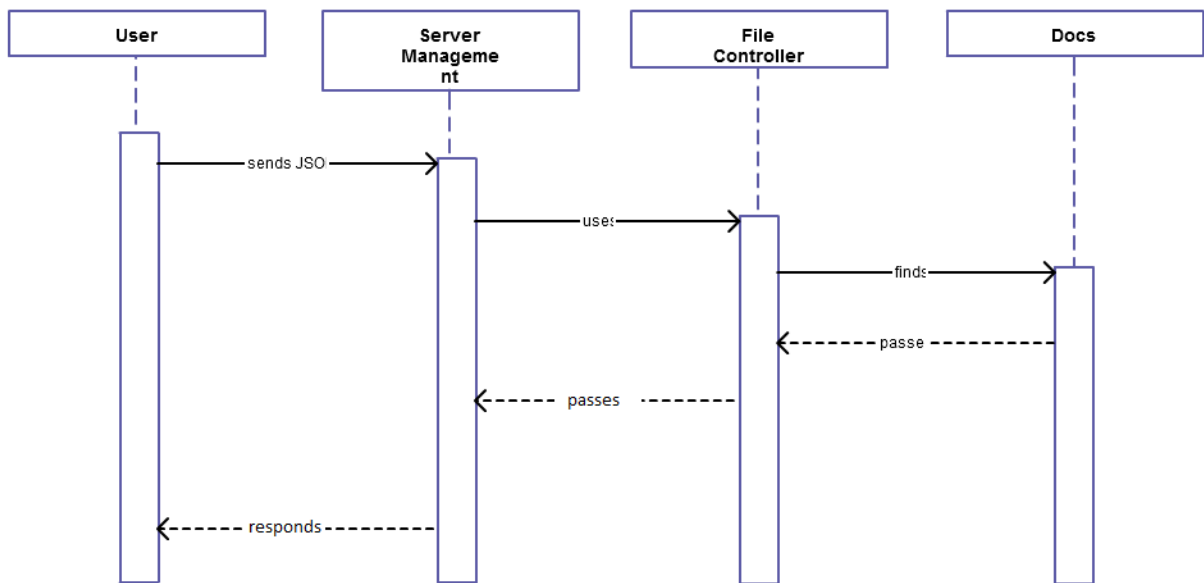


Figure 14 - Sequential Diagram of Server Use Case 3

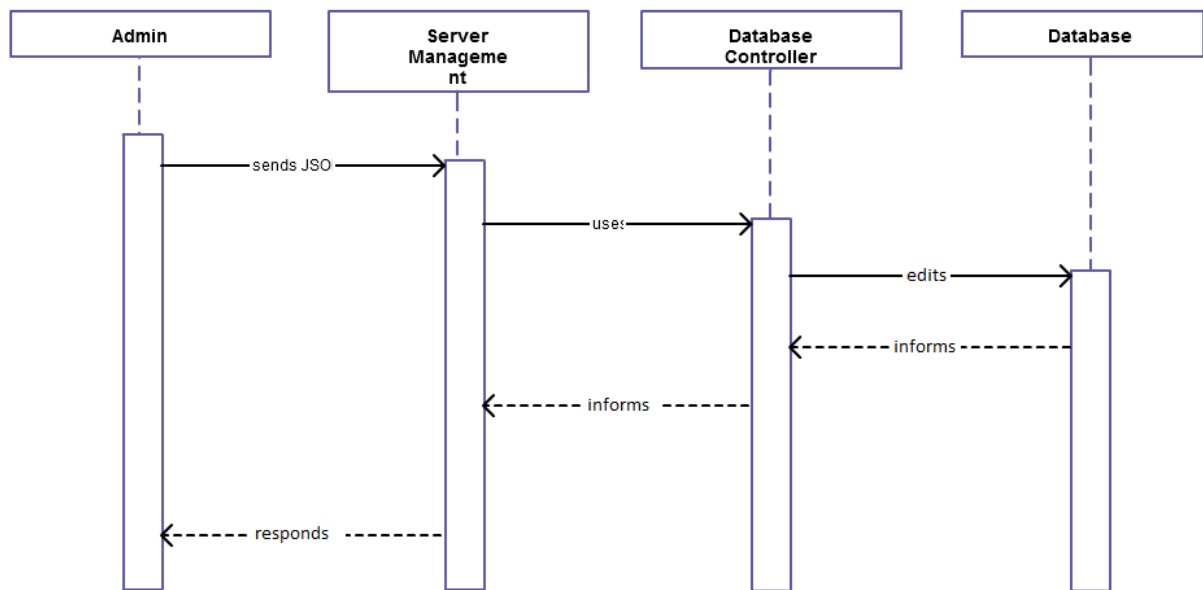


Figure 15 - Sequential Diagram of Server Use Case 4

### 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 administrators. So, client side is also separated to two different sub-systems. As a result, system is separated to three components which are User Component, Administrator Component and Server Component.

One approach may be separating 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 may be separating 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, administrator 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 of the system, 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 13. If given info is correct, user redirects to the design area screen, else a warning is written to the screen.

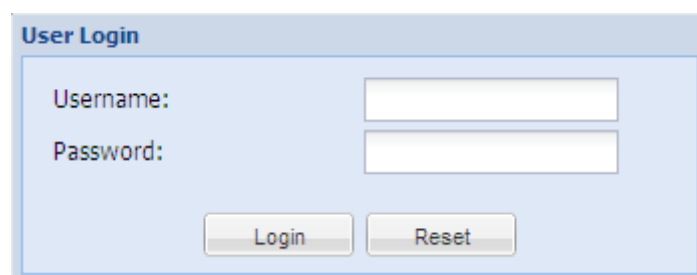
The image shows a 'User Login' window with a light blue border and title. Inside, there are two labels, 'Username:' and 'Password:', each followed by a white text input field. Below the input fields are two buttons: 'Login' and 'Reset', both with a light gray gradient and a thin border.

Figure 16 - User Login Screen

#### 6.2.2. Document Selection Screen

In this screen, which is shown in figure 14, 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.

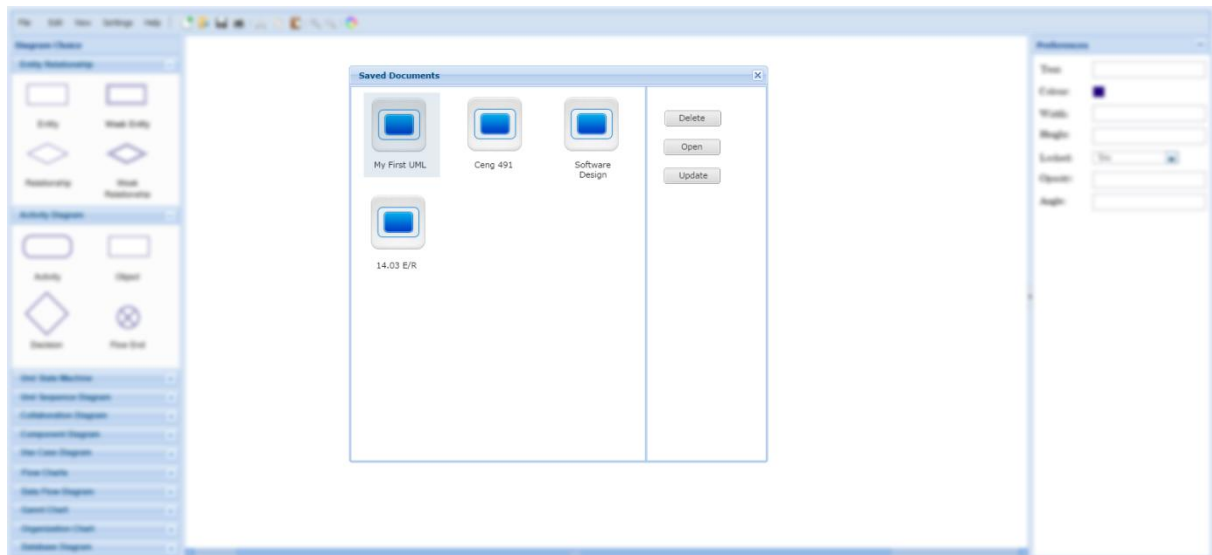


Figure 17 - Document Selection Screen

### 6.2.3. Design Area Screen

Most of the front-end process is taken in this screen which is shown in figure 15. 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.

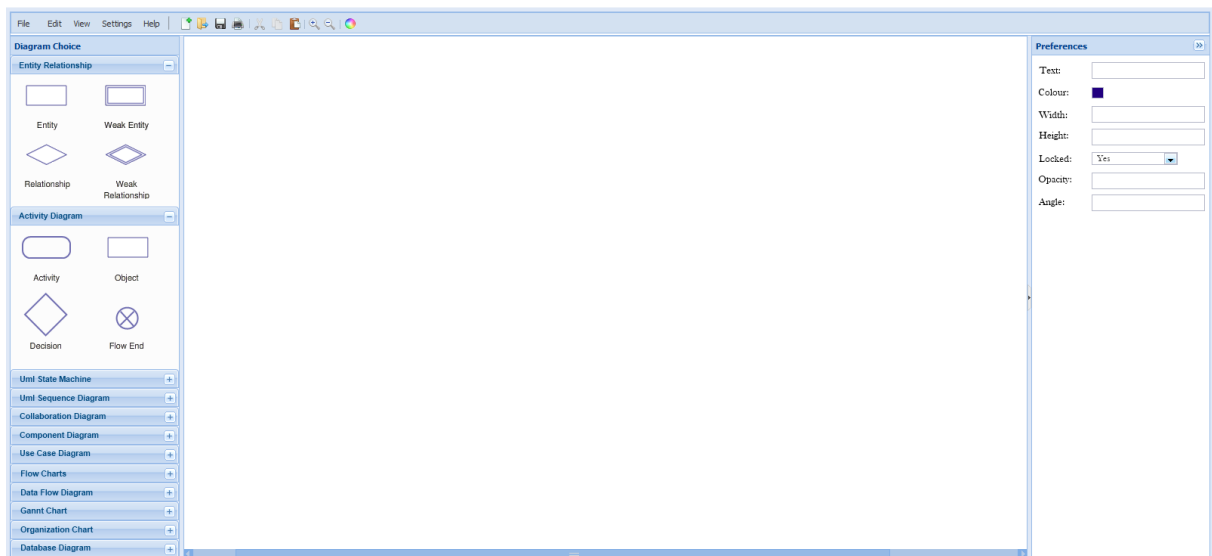
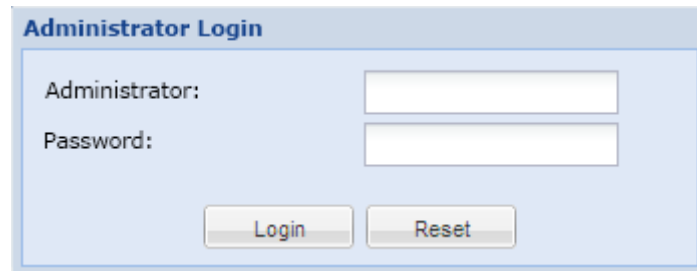


Figure 18 - Design Area Screen

#### 6.2.4. Administrator Login Screen

The first step when administrator opens the iFlowEdit is entering login information in the screen shown in figure 16. If given info is correct, administrator redirects to the administrator area screen, else a warning is written to the screen.

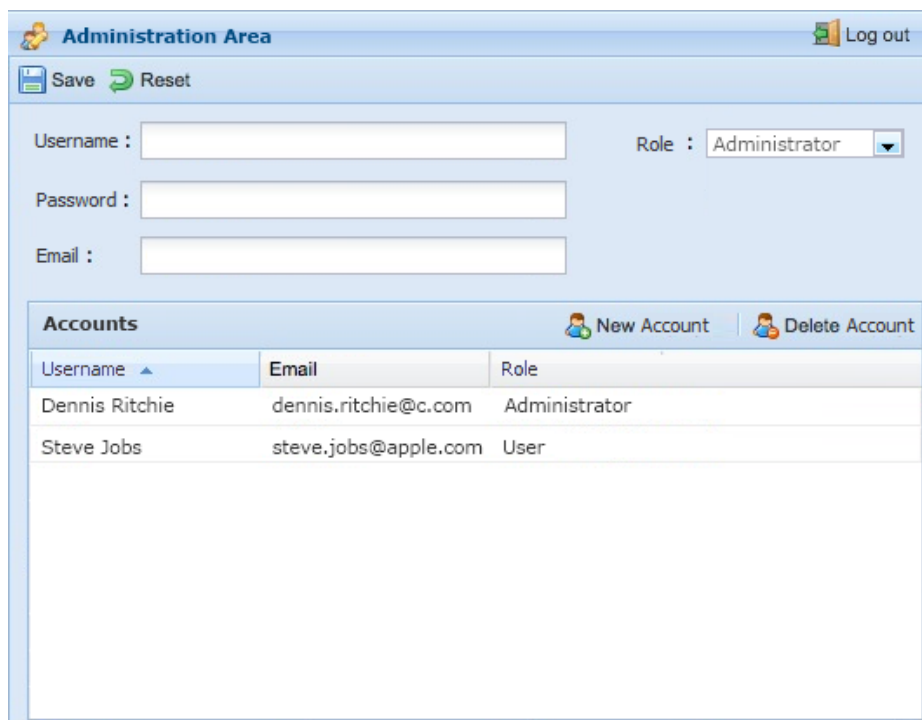


The screenshot shows a login window titled "Administrator Login". It contains two input fields: "Administrator:" and "Password:". Below these fields are two buttons: "Login" and "Reset".

Figure 19 - Administrator Login Screen

#### 6.2.5. Administrator Area Screen

In this screen, shown in figure 17, administrator can add new user or administrator to the system; sees, deletes and updates accounts' details. Administrator can log out via log out button.



The screenshot shows the "Administration Area" screen. At the top, there is a "Log out" button. Below it, there are "Save" and "Reset" buttons. The main form contains three input fields: "Username :", "Password :", and "Email :". To the right of these fields is a "Role :" dropdown menu with "Administrator" selected. Below the form is a table titled "Accounts" with columns "Username", "Email", and "Role". The table contains two rows: "Dennis Ritchie" with email "dennis.ritchie@c.com" and role "Administrator", and "Steve Jobs" with email "steve.jobs@apple.com" and role "User". To the right of the table are two buttons: "New Account" and "Delete Account".

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

Figure 20 - Administrator Area Screen

### 6.3. Screen Objects and Actions

The interaction is shown in figure 18 from the beginning of the administrator log in and user log in.

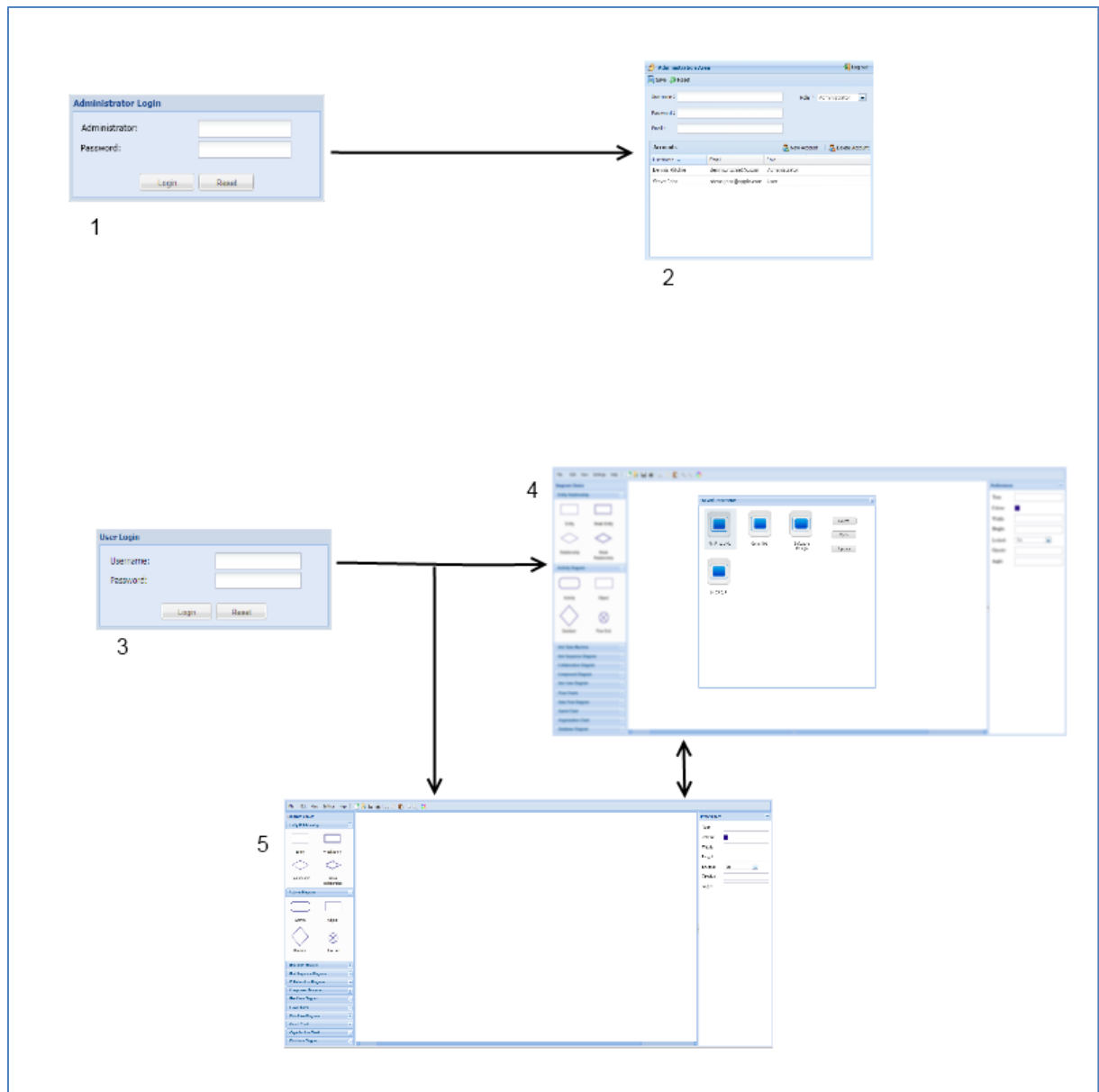


Figure 21 - Screen Objects

The interface denoted with 1 is administrator login; administrator clicks the login button and then goes to administrator 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. Libraries and Tools

As it is mentioned before, the technologies that will be used are HTML5, JavaScript, CSS, Ext-JS, ProcessingJS, XAML and JSON. HTML5, CSS, JSON and XAML affect functionality indirectly, while JavaScript, Ext-JS and ProcessingJS affect functionality directly.

### *The ones that have indirect effect*

HTML5 will be used to create everything on the user GUI and administrator GUI. Stencil set, work flow, 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.

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

XAML will be used to store the information of the work flow. It will be used with WF.NET.

### *The ones that have direct effect*

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.

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.

## 8. Time Planning

### 8.1. Term 1 Gantt Chart



Figure 22 - Term 1 Gantt Chart

### 8.2. Term 2 Gantt Chart



Figure 23 - Term 2 Gantt Chart



## 9. Conclusion

This document is a detailed initial guide for process of the project that will be done in the future. In the document, structural and architectural design of the project is explained. Component and sequential diagrams of the modules of the system are explained in details. Tools and libraries that will be use is mentioned, however some of them may be change according to the needs of the system in the future. Tools and libraries will be decided just before the implementation of the system. Any change of them will be explained in details. After this documentation, Detailed Design Report of the project will be written, which contains more detailed design of the structure of the system.