

Fall 2016

MABIC: Mobile Application Builder for Interactive Communication

Huy Manh Nguyen

huymanh.nguyen378@topper.wku.edu

Follow this and additional works at: <http://digitalcommons.wku.edu/theses>



Part of the [Databases and Information Systems Commons](#), [Information Security Commons](#), and the [Software Engineering Commons](#)

Recommended Citation

Nguyen, Huy Manh, "MABIC: Mobile Application Builder for Interactive Communication" (2016). *Masters Theses & Specialist Projects*. Paper 1747.

<http://digitalcommons.wku.edu/theses/1747>

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact topscholar@wku.edu.

MABIC: MOBILE APPLICATION BUILDER FOR INTERACTIVE
COMMUNICATION

A Thesis
Presented to
The Faculty of the Department of Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Computer Science

By
Huy Manh Nguyen

December 2016

MABIC: MOBILE APPLICATION BUILDER FOR INTERACTIVE
COMMUNICATION

Date Recommended 11/18/16



Dr. Guangming Xing, Director of Thesis



Dr. Zhonghang Xia



Dr. Huanjing Wang



11/21/16

Dean, Graduate School

Date

ACKNOWLEDGMENTS

Firstly, I would like to thank my graduate advisor, Dr. Guangming Xing. It is and always will be a great pleasure working under the guidance of Dr. Xing. His office is always open whenever I ran into trouble or had questions about the project. The experience and knowledge that I learned from him are invaluable. I would like to express my gratitude and appreciation to Dr. Xing for his immense trust and patience. He consistently gives supports guide me in the right direction. This thesis would not have been succeeded without him.

I would like to thank Dr. Zhonghang Xia and Dr. Huanjing Wang for their recommendations and valuable time that helped me improve this Thesis. I would like to thank Dr. James Gary for giving me an opportunity to be a Graduate Assistant. This job trained me about the responsibility and self-discipline when working in a professionalism environment.

I want to say thanks to all my friends at WKU, my friends in Vietnam for their endless support.

I would also like to extend my deepest gratitude to my family, especially my mom. I would like to say thanks for her everlasting love and inspiration for me to succeed. Without her, I would not have a chance to study at Western Kentucky University and got my Master Degree completed.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION	1
CHAPTER 2: BRANCHING LOGIC.....	5
CHAPTER 3: WORKFLOW MANAGEMENT SYSTEMS	6
1. INTRODUCTION	6
2. MODEL	7
CHAPTER 4: COMPONENT-BASED SOFTWARE.....	9
CHAPTER 5: MABIC APPLICATION	11
A. USER INTERFACE	11
1. SERVER SIDE.....	11
1.1. Main Menu.....	11
1.2. Apps Interface.....	12
1.3. Data Collector	14
2. CLIENT SIDE	18
2.1. Main Interface.....	18
2.2. Sidebar	19
2.3. Setting.....	19
2.4. My Profile	19
2.5. My Apps.....	20
B. ARCHITECTURE	22
1. SERVER SIDE.....	22
1.1. Java Server Faces & PrimeFaces	22
1.2. MySQL Database.....	23

1.3.	Apache Tomcat and Maven	24
2.	CLIENT SIDE	24
2.1.	HTML and JavaScript.....	24
2.2.	jQuery Mobile.....	26
2.3.	Apache Cordova.....	27
C.	WORKFLOW	28
1.	SERVER SIDE.....	28
1.1.	Login.....	30
1.2.	App and Data Collection Creation.....	31
1.3.	Schema Generation	31
2.	CLIENT SIDE	36
2.1.	Login.....	36
2.2.	App and Data Collector	39
2.3.	Profile Update	44
2.4.	Local Notification	46
	CHAPTER 6: CONCLUSION.....	48
	REFERENCES	50

TABLE OF FIGURES

Figure 1: Main Menu Webpage	12
Figure 2: Interface of Apps Webpage.....	13
Figure 3: A dialog of app creation	14
Figure 4: Edit Schema View	15
Figure 5: Setting Operations and Targets	17
Figure 6: An architecture of the server	22
Figure 7: An architecture of the client	25
Figure 8: The Server Workflow	29
Figure 9: Pre-defined roles using Spring Security Framework	30
Figure 10: A view of existing apps	31
Figure 11: Condition Editing View.....	32
Figure 12: Information of An Action.....	34
Figure 13: Client Workflow	37
Figure 14: An example of request body.....	38
Figure 15: A request result of an app.....	40
Figure 16: Example of a question	41
Figure 17: An Example of Client's Request to Server	42
Figure 18: History View of Data Collector.....	43
Figure 19: Result of profile's request	45
Figure 20: Update Profile Request.....	46
Figure 21: A Local Notification View	47

MABIC: MOBILE APPLICATION BUILDER FOR INTERACTIVE COMMUNICATION

Huy Manh Nguyen

December 2016

53 Pages

Directed by: Dr. Guangming Xing, Dr. Zhonghang Xia, Dr. Huanjing Wang

Department of Computer Science

Western Kentucky University

Nowadays, the web services and mobile technology advance to a whole new level. These technologies make the modern communication faster and more convenient than the traditional way. People can also easily share data, picture, image and video instantly. It also saves time and money. For example: sending an email or text message is cheaper and faster than a letter. Interactive communication allows the instant exchange of feedback and enables two-way communication between people and people, or people and computer. It increases the engagement of sender and receiver in communication.

Although many systems such as REDCap and Taverna are built for improving the interactive communication between the servers and clients, there are still common drawbacks existing in these systems. These systems lack the support of the branching logic and two-way communication. They also require administrator's programming skills to function the system adequately. These issues are the motivation of the project. The goal is to build a framework to speed up the prototype development of mobile application. The MABIC support the complex workflow by providing conditional logic, instantaneous interactivity between the administrators and participants and the mobility. These supported features of MABIC improve the interaction because it engages the participants to communicate more with the system. MABIC system provides the mobile electronic communication via sending a text message or pushing a notification to mobile's device. Moreover, MABIC application also supports multiple mobile platforms.

It helps to reduce the time and cost of development. In this thesis, the overview of MABIC system, its implementation, and related application is described.

Chapter 1: Introduction

The growing availability of Web services and mobile technology significantly brings new opportunities to enhance effective communications: between people and people as well as people and computer systems. Unlike the traditional one-way communication like reading a book and watching TV, interactive communication is dynamic, two-way communication [7]. Nowadays, interactive communication is widely used in targeted marketing where personalized experience could increase the participation and satisfactions of the consumers.

The motivation of this project is to build a framework to support fast prototyping of health and wellness applications. One of the biggest challenges in the traditional care and wellness industry is the lack of interaction between program administrators and participants. For administrators, they often want to track the health condition of participants frequently. This benefits for both administrators and participants because participants will receive essential information, good advice or treatment immediately from administrators. Moreover, by collecting the data from the participant instantly, these data can be processed by the system in real-time or be analyzed later for statistics, education or research purpose. The system also can respond feedbacks to participants rapidly. A well-design interaction system not only supports one-way connect with users, but also engage users in communicating back to the system.

There are some software systems such as Research Electronic Data Capture (REDCap) [10] or Taverna [27] that try to solve these problems. Even though these systems have some advantages, there are still flaws exist in these systems. The REDCap project provides a good interface for collecting data from the users. Forms can be easily

created by the administrators, and the forms are then presented to the participant users for data collection. All the information will be saved into databases and can be further processed to present to researchers. Three major drawbacks of REDCap [10] are:

- The lack of supporting branching or conditional logic, all the question in the form is static.
- REDCap also does not support real-time communication with its users and cannot engage the participants using current technologies like SMS or push notifications.
- It requires continuous access to the Internet, thus limits the accessibility to the users.

Another related system is Taverna, which has a suite of workflow tools. It is designed to combine distributed Web services and local tools into complex analysis pipelines. Most of the drawbacks of REDCap that are mentioned before are fixed in the Taverna software. For example, Taverna supports branching logic by providing the workflow management system. Besides, Taverna support deployment on the local machine so it can be used without the internet by using Taverna workbench, or it can still be used in standard ways, such as by a browser.

However, both Taverna and REDCap system still do not support the instantaneous interactivity between administrators and participants or automated data collection for users. Also, both do not provide the appealing interface that attracts users. Still, these features are the most important in an interactive communication system.

MABIC web services and mobile application is developed to resolve these problems. MABIC web service is designed as a platform for the interactive

communication between administrators and participants in the system. These are a few significant advantages of using MABIC system:

- Firstly, MABIC supports branching logic. The admin can not only create questions but also to set conditions for activating a question or a group of questions. Instead of moving linearly from one question to another, MABIC supports dynamic changes. Now, from one question, depends on the answer to that question, different questions will be presented. This feature is very useful; it helps administrators to create various complex forms. Besides, data collection also benefits from this feature. It allows administrators to create a form that is only collecting desired data from participants.
- Secondly, MABIC system supports automated data collection. The administrator can create new forms without spending too much time because every form can be reusable. Also, the participant's collected data will be automatically sent to the server and stored safely there. MABIC system also provides tools for analyzing these data collections and then gives feedbacks immediately to the users.
- Additionally, MABIC system interface is very simple but appeal, it is very easy for administrators to create a schema without worrying about what programming languages of the system are.
- Another advantage of MABIC system is that it supports instantaneous interactivity and mobility. Also, MABIC supports multiple mobile platforms so that participants can easily respond at anytime and anywhere. Offline forms also are supported; participants do not need the internet to give the response. Instead of that they can go ahead and select the answer and when whenever the internet

connection is available, all data will automatically be synced with the server immediately. Besides, MABIC system can also send reminders to users to alert them about new forms or require participants to input data for the data collection.

Overall, MABIC system supports branch logic for creating new forms, schemas or surveys, and instantaneous interactivity. Administrators can easily set up automated reminders or notifications to encourage participants engaging with the system. Moreover, mobility is another advantage of MABIC; participants can respond to the system at anytime and anywhere by text message, email or through the mobile application. These benefits of MABIC system can improve the interactivity communication between administrators and participants significantly.

Chapter 2: Branching Logic

Branching Logic is used when form makers want to send the responses of users to different paths of the survey. Branching logic is like the chosen answers on the adventures in the role-playing games [18]. The differences of surveys can be determined based on the condition of the answers, variables of the questions or the combination of multiple variables. Different paths will be taken depending on how users select their answers for certain questions. The conditions can be either simple or complicated. For example: if the reply to a question is A, next question will be X, or if the answer is B, next question will be Y. Moreover, the conditions can be a complex combination of multiple choices, e.g. if the answer of question X is A and answer of question Y is not B, question Z will be loaded, or if the condition is not matched, the survey will be terminated.

Branching logic forms follows flowing order from top to bottom and left to right. Responders must answer the first question and go down to the next one. If the condition of a branch is met, next questions will be loaded following that branch until they reach a condition that they do not satisfy. A branch is also ended when it reaches to the end. Branching logic is applied to workflows by allow participants to follow one path or another path based on the actions of the application. For example: if responders do not input data on time, a reminder will be automatically sent to the device to notify them; otherwise, that reminder will never be fired off.

Chapter 3: Workflow Management Systems

1. Introduction

A workflow is a particular kind of process that from an activity, it can jump to another activity; or transform from a state to a different state per a set of conditions or procedural rules specified in the system. The specification of the rules for transitions is managed by Workflow Management Systems (WfMS).

Even though definitions for a WfMS can vary significantly by different authors and the functionalities supported in workflow products are different from one to another, the WfMS standard definition can be clarified as: "Workflow Management Systems consists of a sequence of activities, the input, and output. Depending on the system, the input, and the output can serve different purposes. An activity is a distinct process stage in the system that can be performed by a user or an automated system. Each activity can include one or more task at the same time. A collection of a task is called worklist, and each task is described as a work item. WfMS supports creating and managing work item and present it to the user." [8].

WfMS supports creating workflows, each workflow created is called an instance. This instance works as a logical or generalized model of WfMS [12]. The job that to be executed when WfMS initializes a new workflow is often called a case. Depends on information of cases, WfMS will give different data to different cases. Therefore, each case will be handled differently by the WfMS. However, multiple cases are not necessary to be orderly processed but can be operated simultaneously. Due to the differences of data associated with various cases, each case can process through different paths. These processing mechanisms are controlled by the workflow system.

One of the advantages of WfMS is that it can separate the logic of workflows and the logic of applications [13]. This can help to push business process out of the application. The benefits of a WfMS can be illustrated as a benefit of Database Management System (DBMS) to data. A DBMS helps developers not to worry about the management of physical data in the system. Similarly, a WfMS can reduce the complexity of processing task in the system. By using WfMS, developers do not need to be concerned about managing the flow of the data and controller between the activities or tasks. It also will improve the flexibility and integration of applications.

2. Model

WfMS model is used to illustrate relationships between WfMS system, users, and other software systems. There are five perspectives of the WfMS model: functional, behavioral, informational, operational and organizational perspective [15].

- The functional perspective: this view indicates what workflows will do. This perspective breaks down workflows to a small task that can be assigned to users or automated computer.
- The behavioral perspective: this view specifies conditions or procedural rules for each work item in the workflow. Each activity in the workflow will be associated with a pre-defined time that explains when will these events be triggers. It also shows what will happen after events are triggered. This perspective indicates that WfMS control the timing of when tasks will be executed.
- The informational perspective: this view related to data that associated with each case of the workflow. It describes what data is consumed and what workflow will

produce. In general, it can be understood as what is the input and output of the workflow. Associated data can be documents, files, forms or databases that store important information of the application.

- The operational perspective: this view explains how WfMS implement workflow tasks. Based on the conditions that are set up by the behavioral perspective, operational perspective will provide necessary tools and applications to complete the tasks.
- The organizational perspective: this aspect is related to the people that perform the tasks. Each person will be assigned to a role. A role is a collection of tasks and responsibilities that belong to the users. A user needs to be authenticated and authorized to check if he/she has the rights to execute the tasks. The organizational perspective purpose is to provide the list of roles, users authentications, access authorizations, workflow documents and workflow manuals. It also provides the list of users, teams, groups and collection of software applications that are applied in the workflow.

Chapter 4: Component-based Software

MABIC system works as a component-based software, just like the middleware between servers and clients. It provides users with a simple interface and a framework that helps users easily to create branching logic workflows.

According to [16], the definition of a software component is "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".

As clarified in the definition, a component is independently deployed, and it is separated into another component. However, it cannot be deployed alone, but rather be deployed with another component. Moreover, the implementation of the component is enclosed, it means that it cannot interact directly with outside environments. Instead of that, the interaction occurs through a well-designed interface. There are six essential characteristics of a component:

- Reusability: components are designed to be used again in many different applications or a particular program.
- Replaceable: a component can be replaced by another similar component.
- Not context specific: one component can be used in different context, instead of just focusing on an environment.
- Extensible: the component can be extended to provide new features.
- Encapsulated: component interacts with environments through an interface, therefore it does not expose internal details of the component.

- Independent: each component is designed to have minimum impact or dependency on another component.

Chapter 5: MABIC Application

A. User Interface

1. Server Side

1.1. Main Menu

MABIC user's interface provides a lot of methods and menus that help administrators easily to interact with the program. Figure 1 shows a typical main menu view of MABIC application. From the main menu, administrators can access multiple functions of the application:

- Invitation: administrators can generate and send an invitation to the participants.
- Comm Templates: this is used for generating templates. Later, these pre-defined templates will be added to the action's condition.
- Custom Attributes: besides pre-defined attributes, administrators can also create a new custom attribute.
- Org Profile: this is the organization information of current users.
- Apps: this is the app management. Administrators can create, modify or remove apps and schemas.
- Users: this is user management. A new user can be generated, modified or deleted by privilege user e.g. administrators.

- Profile: administrators can view and edit their profile information.

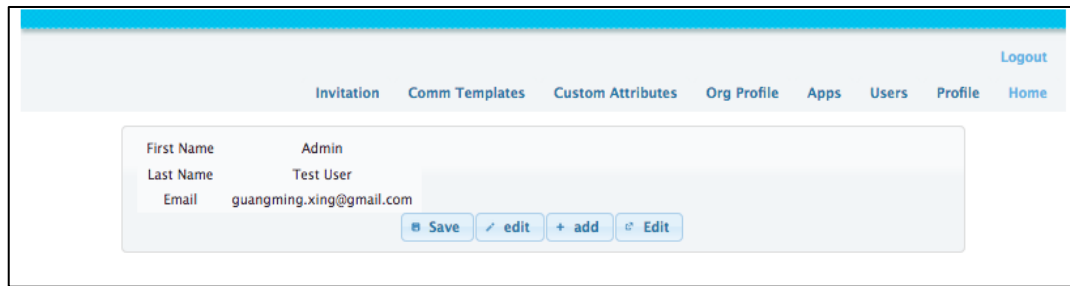


Figure 1: Main Menu Webpage

1.2. Apps Interface

“Apps” interface contains a list view that displays current apps that are created by administrators. Figure 2 shows a view of the app interface. Each app contains four different functions:

- Edit App: this is used for modifying app’s information such as app title, app’s owner information.
- Edit Schema: administrators can build a new schema by selecting Edit Schema of an app.
- Report: a report that contains all information about the app will be generated. This report can also be exported to different formats e.g. pdf or word file.
- User Resp: this will generate a report of participant’s responses for an individual app. This report can also be exported.

The app is the main content of the system; administrators can generate an app and then construct it as they desire by creating multiple question items. The app holds the content that administrators the server want to send to participants on the client side. These contents can be a collection of questions, images, reports or reminders.

Apps can be generated by clicking the tab Apps on the main menu and then choose the Create button. When creating a new app, there are some properties that administrators need to. Figure 3 shows a dialog of creating an app.

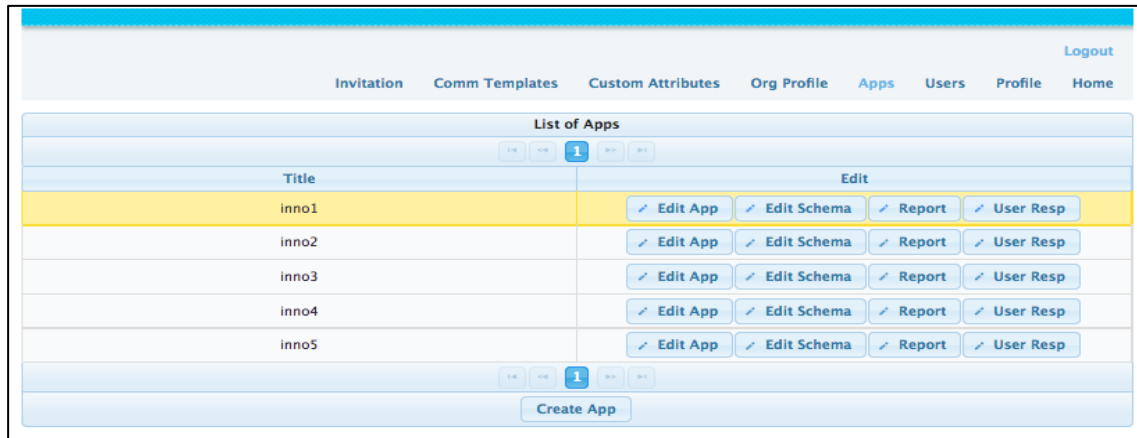


Figure 2: Interface of Apps Webpage

- Title: this is the display name of the app such as inno1, inno2, etc.
- Email: this is the email of the creator of the app, it can be used as contact information or local variables when creating conditions in schema section.
- Phone: the phone number of the app owner. It can also be used in local variables in schema section.
- Type: the type of the app. It can be simple or long term type.
- Timeout: the duration time that app can exist before it is being removed from the server (0 means unlimited time).
- Upload Schema: admins can upload schemas (JSON format) that are created before. Because the schemas are reusable, this will reduce app creating time.

New question item can be created by clicking on Edit Schema button. A list view that contains a list of current items that are created previously for the app will be displayed. There will be a few options to interact with the item such as creating a new

item or editing the current one. After modifying an item, admins need to save the content of the current item by clicking Save button. If there are many items are generated, a new page list will be created. Figure 4 below shows an example of Edit Schema view.

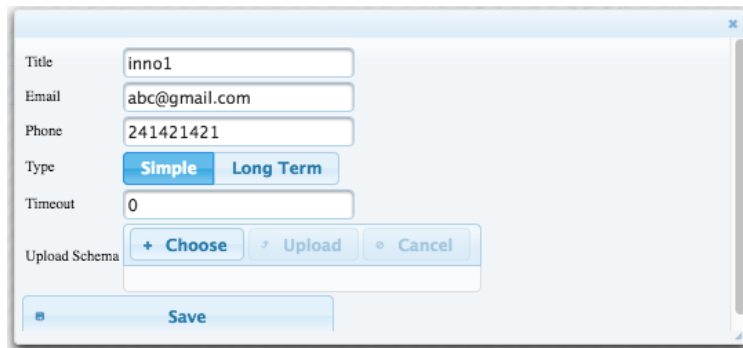


Figure 3: A dialog of app creation

1.3. Data Collector

Data collection module can be accessed by expanding “Data Collection” field in the main app. Each module is created by administrators to collect data from participants. Admins will create a data collector and then specify the content such as mood, blood pressure or body temperature, etc. Received data from participants will be sent back to the server and later be analyzed for different purposes. Administrators can create a new data collector or modify each module by using the Edit button. These are some required attributes that need to be defined for an app:

a) Properties

- Rolling Start: this property has two value, true and false. If it is true, participants must enter values for the full period; from the initial day to the complete day, e.g. 30 days. Otherwise, participants can enter values from the current day to the ended day that was established up the system.
- Num of days: the number of days that participants need to enter the value e.g. 15 days.

- **Markers:** Markers indicate the time of the day that data will be collected. Markers can be a list of the string such as morning, afternoon, evening, etc.
- **Reminder Frequency:** the reminder is used for notifying participants on the mobile device (as known as local notification). The local notification helps to remind users to response to the question or input data values. This attribute indicates how frequency the local notification will be triggered. It also can be combined with the reminder period to compute when notifications are triggered.
- **Reminder Period:** this attribute indicates the period that notifications will be repeated; its value can be a day, week or month. It combines with the reminder frequency to compute the precise timing that triggering local notifications when.

b) Variables

- **User Attributes:** this contains attributes that belong to the owner of the schema such as first name, last name, email, phone, age and day of birth.
- **Defined Attributes:** the administrators can create new attributes or remove the existing attributes for the individual app.

c) Items

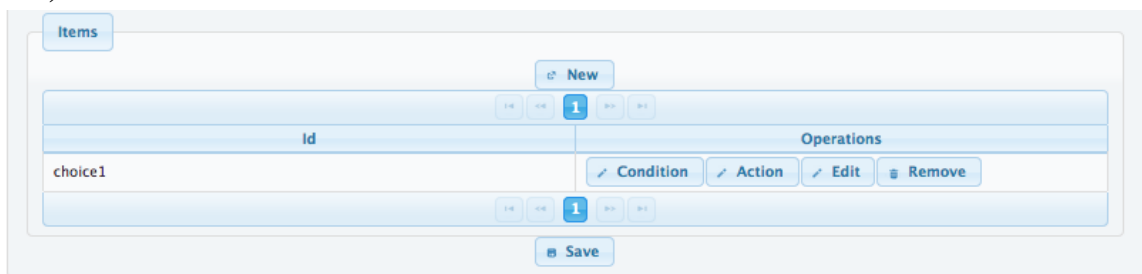


Figure 4: Edit Schema View

The administrators can create questions or items for a schema. The type of a question item can be the choice, check, range, report or group type. Group type is a special type that contains multiple other different question types. Administrators can

create, modify or remove any items. Also, they can also add the conditions and actions for each item.

- Item: there are five question types: choice, check, range, report, and group. There are some basic fields for an item that need to be specified: id, title, instruction, description, isStart, and isEnd.
 - Choice: with this type, each question item will have multiple selections. Each answer will contain a label, value, and description. The answer of this type is unique; it means that there is only one correct answer for the question.
 - Check: check type has same properties as choice type. However, the answer is not unique; it can be a combination of the individual answer.
 - Range: range type contains only two values that indicate the maximum and minimum value of the answer.
 - Report: report type contains a text box that allows participants to enter required data.
 - Group: this is the combination of all previous types. For example, group type can be a combination of choice and check type or report and the range type.
- Condition: item's conditions allow administrators to indicate which question will be loaded after the previous one was answered. Each condition will be represented as a node. Different conditions can be combined to create a complex condition. Each condition is specified by three properties as in Figure 5: operations, operands, and targets.

- **Operation:** the operation specifies the comparison that will be used to indicate the condition such as equal (=), less than (<), larger than (>), less than or equal (<=), greater than or equal (>=), not equal (!=) and at (@) operation.
- **Operand:** this indicates whom will perform the operation. Operands can be item id, local variables or global variables. This information can be retrieved from the app.
- **Target:** this is the target of the operation. The target will be entered as this format @value@. This value can be a number or a string. When the operation matches with the target, this item will be processed.

Operation:	= > < >= <= != @
Operand	CollectorChoice1
Target	@1@

Figure 5: Setting Operations and Targets

A compound condition is represented as a tree structure, with each node represents a condition. Each condition can be associated with another condition by logical operations AND, OR, NOT. The admins can specify operations between conditions by selecting a condition and then adding an operation.

- **Action:** This is the action that admins want to set when an item appears or disappears. An action is created by specifying these properties:
 - **PrePost:** Pre means before an item appears and Post means after an item disappears.

- Operation: it can be an email, SMS or push. It means that the action can be sending an email, a text message or pushing a notification to participants.
- Recipients: it can be the app, user or listed. The “listed” type is the list of recipients; this is useful for sending multiple recipients.
- CommTemplate: this is body content of the action.
- Subject: this is the subject content of the action.
- Target details: this contains recipients’ information. It can be the email address or the phone number.

2. Client Side

The client side of MABIC application supports multiple platforms such as iOS, Android, Window Phone, etc. This mobile app is developed using web-based technology for the mobile application.

Firstly, participants need to be authenticated by entering username and password in the login page. This login page including two text boxes that requiring participants to input. After logging in successfully, participants can access the main interface of the app. All important functions of the application can be accessed through the sidebar on the left side. This sidebar contains all the functions of the application including “Log Out,” “Setting,” “Profile,” “App” and “Data Collector.”

2.1. Main Interface

After participant’s login are approved successfully, a default app will be loaded. This default app is an app that is established by the administrator; it will automatically load when the application starts. From the main interface, participants can access the

sidebar by clicking on Menu button on the top-left of the screen. Here, participant's information such as first name and last name will be presented.

2.2. Sidebar

The sidebar includes several buttons; each button is used to access different methods. This Menu button will be available most of the time in the application so that participants can easily change to another function.

- In the sidebar, participants can get to the setting page by clicking on the Setting button.
- Participants can also update or modify their information by clicking on My Profile button
- They can also start another app or data collector by selecting the app button.

2.3. Setting

Reminders for each data collector can be accessed on the setting page. The current setting page allows participants to toggle on and off reminders. If an option for an item is turned off, there will be no local notification activated to remind participants to input required data that need to be collected. However, if this reminder is on, at the specific time of day, this reminder will check if the participant has already entered a value or not, then it will trigger the local. Moreover, if the reminder is on, the application will repeatedly check several times a day or week to keep notifying.

2.4. My Profile

Participant's information will be displayed on My Profile page. This info can be accessed by selecting My Profile button on the sidebar. Participant's information contains first name, last name, email, phone, password, age and date of birth.

All user's info will be presented directly on the page, excluding the password. The password cannot be revealed because it is sensitive and require to be protected. The other information can be updated or modified by changing the value in the text box. After new value is entered; by clicking the Update button, all new info will be saved. The Profile page will be refreshed and display updated info of the participants.

2.5. My Apps

The app and data collector can be accessed through a list of buttons on the sidebar. This list is dynamic; it depends on the data that retrieved from the server side. In My Apps section, there are two different kinds of app, the main app, and the data collector.

- The main app is represented as a color button without the arrow icon. Each app is followed by a list of data collectors. It means that app is the parent app of those data collectors.
- A data collector is a simple white background color button with the arrow icon on the left side of the button text.

a) The App

When participants click on an app, the app will be loaded and the first question will be loaded and displayed on the screen. Participants need to answer each question by selecting the choice they want. If participants do not answer, they cannot go to the next question. An alert will be displayed to inform participants selecting an answer first. For example: if the question type is a choice, an answer needs to be selected before going to the next question; if it is a check type, at least one answer must be selected. After picking

an answer, they can click on Submit button and go to the next question. By submitting answers, all data will be saved sent to the server for analysis purpose.

Participants continue to answer the list of the questions that loaded from the server until all questions are answered, or there are no more questions; then the app will be completed.

b) Data Collector

For the data collector, the interface is similar to the main app; a question will be displayed, and participants need to input required values for that question. However, unlike the app, there will be no question is loaded. Instead of getting next questions, an alert will be displayed to notify that these data are already recorded.

Participants will input required values one time or multiple times per day or week. This recorded information will not only be saved in the local device but also be sent to the server. After the data is recorded, these history data of the app will be updated. History data can be accessed by clicking the History button on the top-right side of the application. These data are shown as an ordered list, each answer for the question and the date that these answers are inputted.

There is also a synchronous mechanism that automatically updates data between the server and the client. This kind of mechanism works based on the timestamp value of which data on server or client is more updated.

B. Architecture

1. Server Side

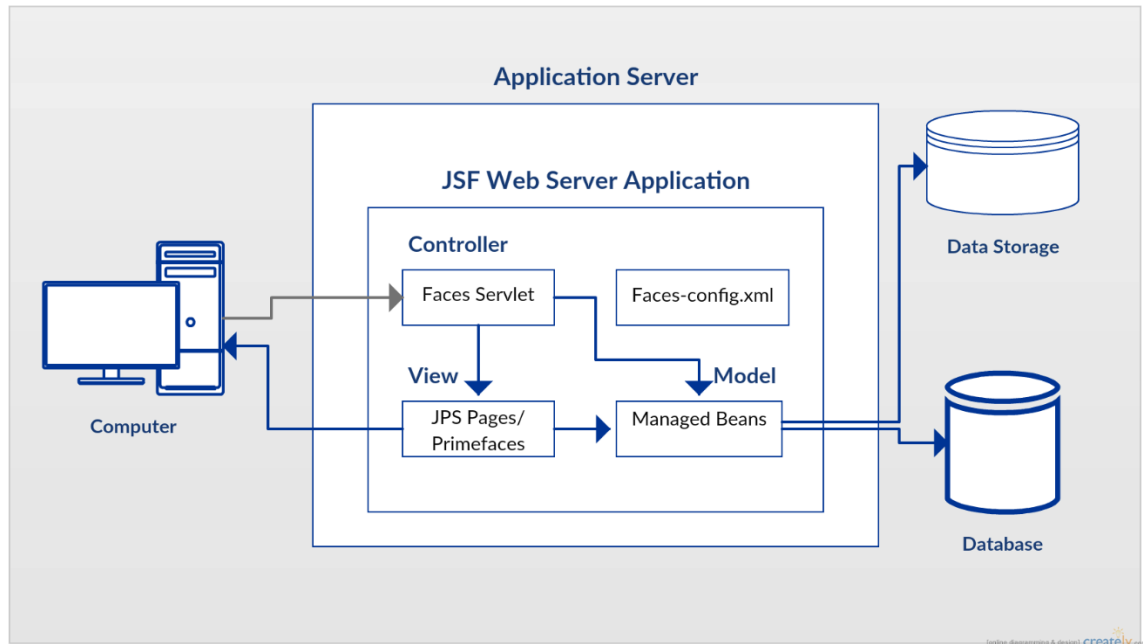


Figure 6: An architecture of the server

1.1. Java Server Faces & PrimeFaces

The MABIC server is developed using Java Server Faces (JSF) technology. JSF is a Java framework for building web applications. The development process will be simplified because the JSF provides a component-based approach to designing the web interface. The developers can just simply drag and drop the components to the web page. The JSF also use the Model-View-Controller design pattern to construct the web page. This design pattern makes the administrators can easily to manage the web content because the code of the view will be separated from the logic of the model. And the controller will handle the interaction between the user and the application 53[24].

The MOBIC server also uses the PrimeFaces library for designing user interface (UI). Primefaces is an open source of UI component library for the JSF. PrimeFaces provides a lot of useful resources for the developers. There are several advantages of PrimeFaces to develop a web application [1]:

- Firstly, PrimeFaces supports AJAX (Asynchronous JavaScript and XML).

AJAX is a client-side script that allows the communication between a server and client without refreshing the page. It can improve the server-client interaction to be fast and responsive.

- Additionally, PrimeFaces provides the Prime UI. It is a collection of rich JavaScript widgets that is based on jQuery UI.

By using PrimeFaces, developers can speed up the development progress of the application. It is also straightforward for developers to design the page layout for the web server pages because it is quite fast and quick to build beautiful pages with minimum effort.

1.2. MySQL Database

The MABIC application uses MySQL database for storing the data. MySQL is an open source database management system (DBMS) for managing and connecting databases with the software. A lot of important information in the system such as user, app, template, and the item is stored in the database. The advantages of using MySQL: it supports the scalability, flexibility, high performance, highly availability, secure data protection, open source and lower cost. The primary benefit of MySQL is that it is convenient and easily to manage. MySQL can also be controlled using visual web tools

such as phpMyAdmin. Moreover, MySQL can be run on multiple platforms such as Unix, Window or MacOS; and it is free for personal use.

The server does not require heavy workload on the database. Therefore, MySQL software is a simple solution for this purpose. In the future, if more features need to be developed, MySQL can still be usage for improving the application.

1.3. Apache Tomcat and Maven

For the local development, MABIC application uses Apache Tomcat software. Apache Tomcat is an open source web server that is developed by Apache Software Foundation. Tomcat is used to run the web application on the local host. Also, Tomcat can be easily downloaded and set up from the Apache website. Combining with the Maven plugin, developers can manipulate WAR (Web ARchive) projects into the Apache Tomcat servlet container. Apache Tomcat and Maven can help to simplify the deployment of the web server.

2. Client Side

2.1. HTML and JavaScript

The MABIC application on the client side is developed using web-based technology, HTML, and JavaScript. The advantages of HTML and JavaScript are

- It is a relatively simple, effective and readable programming language, but also powerful. Besides, JavaScript makes HTML pages more dynamic and interactive with the users. JavaScript and HTML can easily make the application to be adaptable with users' requirement.

- Another advantage of JavaScript is the application can be executed on the client's processor instead of the web server. Thus, saving bandwidth and strain on the web server.
- JavaScript is relatively fast. Mostly, the code will be executed on the client's side. Because the task is usually simple, the computation and processing will be completed instantly and does not require many resources from the server.

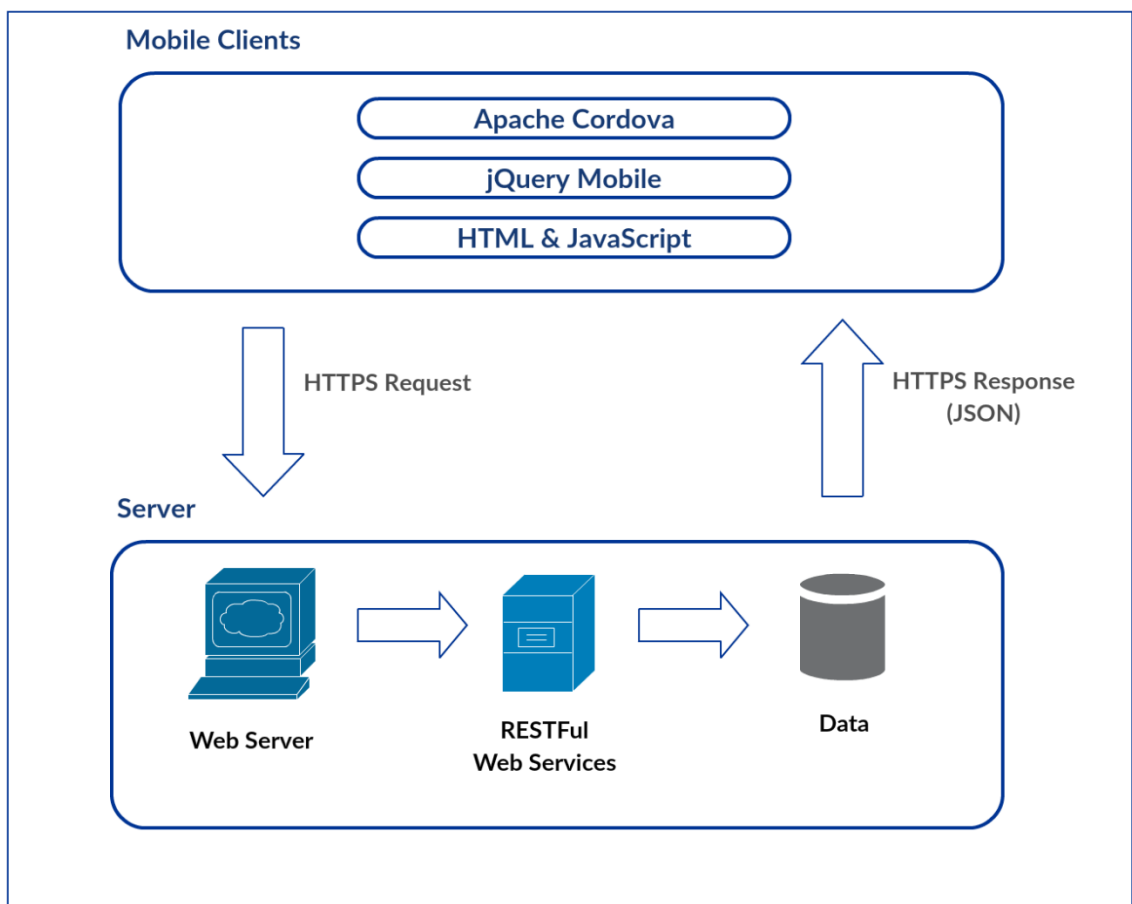


Figure 7: An architecture of the client

2.2. jQuery Mobile

JavaScript contains several libraries supporting mobile application development. And one of the most popular libraries is jQuery Mobile; it is a convenience mobile framework for developing the mobile web-based application.

jQuery Mobile is a JavaScript library that is immensely simplifying the development process. There are several benefits of using jQuery Mobile:

- It is a simple, easy-to-use framework and has condensed syntax. It is much easier when comparing to the standard JavaScript and another JavaScript library. JQuery's syntax is simplified and therefore requiring fewer lines of code.
- jQuery Mobile has a strong open source community, so it is not difficult to look for some appropriate plugins. There are hundreds of pre-written and ready-to-use plugins available for download. Thus this can speed up the mobile application development process.
- jQuery also has comprehensive documentations and tutorials are available on the website, this can help beginners are easy to get ready to develop an application.

One of the biggest benefits of jQuery is that it supports AJAX. jQuery's AJAX call is implemented in a clear and understandable technique. The AJAX is an asynchronous request that is initiated by the browser on the client side. It is an approach to web application development that uses client-side scripting to exchange the data with the server-side. AJAX does not directly require page transition, so the web pages on the client side are dynamically reloaded without refreshing. Therefore, it will not interrupt

the interaction flow of the application. In MABIC application, the AJAX is used for sending the request from the client to the server.

2.3. Apache Cordova

Apache Cordova is a popular mobile application development framework that is owned by Apache. This framework enables software developers, programmers, and designers who want to build applications for mobile devices on multiple platforms by using CSS3, HTML5, and JavaScript. Therefore, Cordova helps programmers focus on developing an application without concerning about platform-specific APIs like in Android, iOS or Windows Phone OS.

There are a lot of benefits when using Apache Cordova to develop mobile applications instead of using the native platform.

- Firstly, Cordova supports cross-platforms; it means programmers do not need to write the code for every mobile system. But the developers only need to write the once and then can deploy it to another platform. It reduces the coding time for developers.
- Secondly, if the developers use the native mobile APIs to develop the application, it is really difficult to maintain the code. For instance, whenever a new version of OS is released, there will be many codes is obsoleted. Therefore, the programmers should spend much time to update the code. This maintenance can be reduced by using Cordova because the Cordova only depend on the web-based APIs in the OS and it is rarely changed.
- Another advantage of Cordova is the programming language. Cordova uses HTML5, CSS, and JavaScript, so it is easier for a web programmer to start

developing mobile applications. Moreover, the beginners do not need to learn multiple programming languages for deploying the application to different mobile platforms.

- There are lots of libraries and examples available for developers to take advantages. This can help to reduce the time and work for developers.

C. Workflow

1. Server Side

The workflow of the server side can be described as four steps:

- Step 1: Firstly, the administrators need to be authenticated and authorized by logging into the system to use MABIC system.
- Step 2: Following, the admins can be able to access all functions of the program by accessing the Main Menu tab bar. It can be creating a new app, generating schemas or deleting an existing user in the system.
- Step 3: After the apps are created, the admins can also modify app information.
- Step 4: For this step, the administrators can create new schemas for the main apps or data collections. The schema is created with JSON format then be saved into the server database and storage. Later, these schemas are retrieved to send information to clients.

Figure 8 is an example the server workflow. Each step of the server workflow is described in details in the next section.

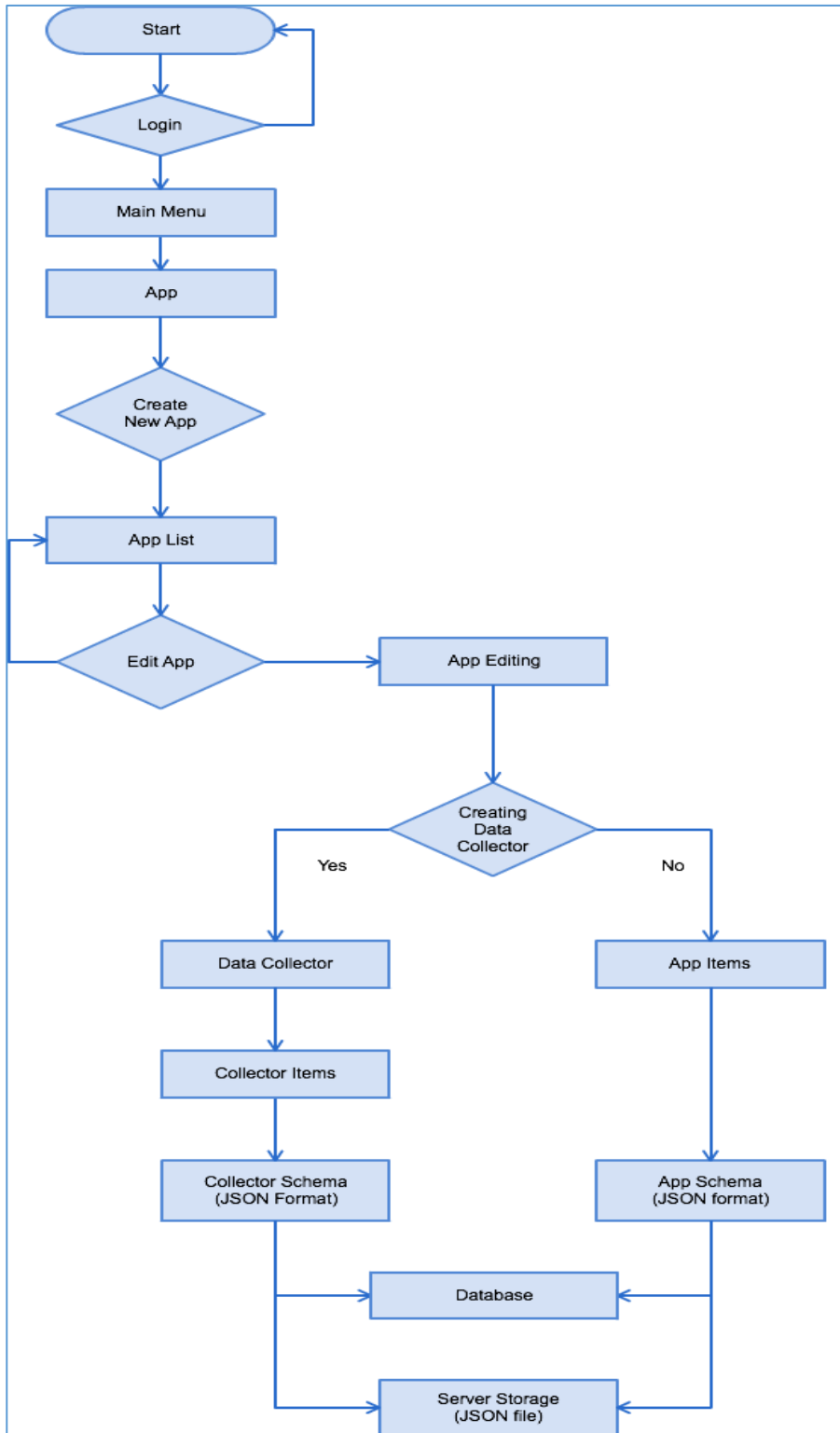


Figure 8: The Server Workflow

1.1. Login

Firstly, administrators log into the system by entering their username and password. The login module uses Spring Security Framework to authenticate and authorize user's information. This framework provides the expression-based access control to handle the login function simply. The advantages of the Spring Security are:

- Firstly, it provides a flexible framework that supports user's authentication and authorization.
- Secondly, Spring Security uses Spring Expression Language (Spring EL) expressions to simplify the configuration of all attributes to authorize user's role. Expression-based access control is introduced in Spring Security 3.0 [6].

Figure 9 shows an example of pre-defined roles that are configured by using Spring Security Framework. In this figure, two roles are pre-defined: the admin and the user. If an account is authenticated as valid a user, he/she can access a web page with the link starting with `/user/`. Also, if this account has the admin role, he/ she can have permission to access a web page with the URL `/admin/`. This intercept URL is parsed from top to bottom. It means that the most specific pattern is standing on top and the catch-all is on the bottom.

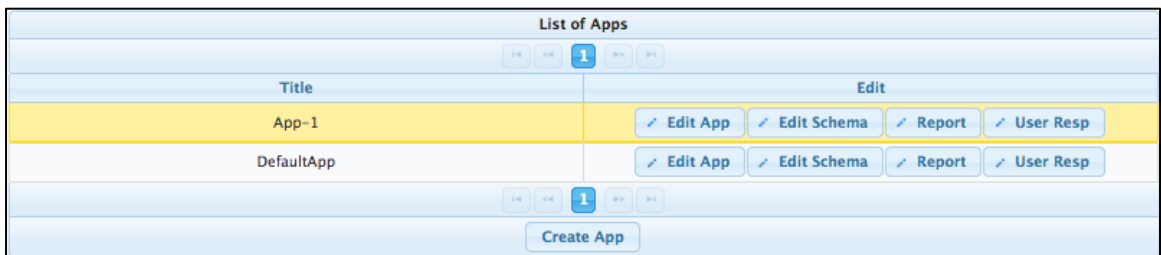
```
<intercept-url pattern="/user/**" access="isAuthenticated()" />  
<intercept-url pattern="/admin/**" access="hasRole('admin')" />
```

Figure 9: Pre-defined roles using Spring Security Framework

1.2. App and Data Collection Creation

To access the app and data collector module, admins must click on “Apps” button in the Main Menu. Then, the apps page will be presented to the screen; this page contains a list of apps that are created earlier. For example: in Figure 10, there are two apps previously existing in the system, “App-1” and “DefaultApp.”

If the administrators want to create a new app, they must click the “Create App” button. Then a new window dialog will show up, and the admins must input all required information and click on the Save button to complete a new app. After an app is created, all app’s data will be saved to the database of the system.



List of Apps	
Title	Edit
App-1	Edit App Edit Schema Report User Resp
DefaultApp	Edit App Edit Schema Report User Resp

Figure 10: A view of existing apps

1.3. Schema Generation

Schema is the most important feature of MABIC application. MABIC system supports branching and condition logic so it can generate the complex schemas. A schema contains a set of questions, conditions, and actions that are prescribed by the administrators. Schema info will be saved in a JavaScript Object Notation (JSON) format. Later, clients will send requests to the server, and these schemas are retrieved and processed. After being processed, the servers will send responses back to clients.

To create a schema, either for the main app or the data collector, the administrators must select the Edit button. Then, if admins want to create main app’s schemas, they should click on the “New” button of the “Items” field. Otherwise, if they

desire to edit schema for the data collector, they must expand Data Collection field and select a Data Collector. Then they must click on the “Edit” button to start editing a schema. An interface of editing schema will show up; it has a similar interface with main app’s schema.

A schema can be generated as these following steps:

- Create an item
- Add condition for items
- Add actions for items

Firstly, admins start creating a new schema by selecting the “New” button of the “Items” field. A first question item needs to be defined in a schema. Later admins can add more question items to the schema. There are five types of question item: choice, range, check, report or group type.

Next, admins can start adding conditions for each item by clicking on the Condition button on the item menu. MABIC system supports branching logic so administrators can be able to set complex conditions for each item; based on the condition, each item will have a different path. A condition editing view is displayed as in Figure 11.

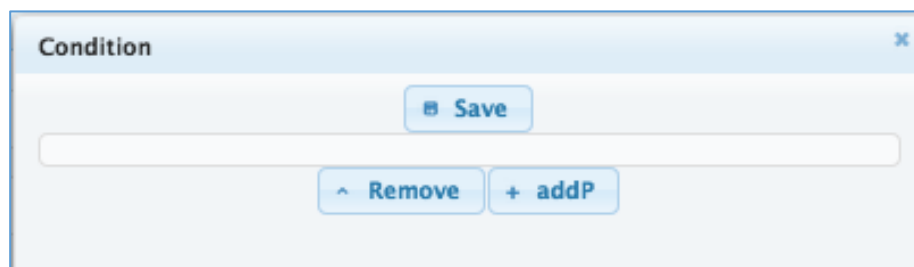


Figure 11: Condition Editing View

When condition editing view is presented, admins must click on “addP” button to add a new condition to the item. Next, a new dialog will show up. This dialog allows

admins to specify the operations and targets for each condition of an item. A condition pointer will be created and added to condition editing view via the addP button.

Each pointer will contain an operation, an operand, and a target. It can be described as if an item “thisId” has a condition with the operand is “itemId,” the operation is “=” and the target is “ansTarget.” If a participant answers the “itemId” question with the answer is “ansTarget,” this answer matches with the condition operation. If the condition is met, the “thisId” item is ready to load. However, a complex branching logic conditions can be created by adding more pointers. Each pointer condition will be associated with logical operation AND, OR and NOT.

The final step is adding actions for schemas. The actions of each item indicate what will occur when a condition of an item is met. For example: if participants get all correct, the server will automatically send a congratulation text message or email to encourage them; or if the participant did not input data more than three days, the system would send a reminder notification to the device.

The action of MABIC system is beneficial for improving the interaction communication between the administrators and the participants. An action will be pre-defined and then be automatically triggered when it’s condition is fulfilled. The administrators define an action by declaring action’s attributes such as pre-post, operation, recipients, template, subject, target details and action delay. Figure 12 shows an example of the action information.

- If the pre-post property is pre, it means this action will happen before a question appears; likewise, post property means that after a question disappeared, this action will be triggered.

- Operation property indicates the type of an action. An action can be sent through email, text message or pushing notifications to local mobile device.
- Depends on the type of an action, others field will need to be specified. For example: if operation type is email, CommTemplate, subject and target details attribute need to be entered. If operation type is a push, recipients must be an app,
- The targets can be individual, app or a list of people and apps. If it is a list, all individual contact will need to be entered.
- The action delay is optional; the administrators can enter information for this field or not.

After fulfilling all required information, the schema will be saved via the “Save” button. The schema’s information process can be described as following steps:

Pre Post	<input type="button" value="Pre"/> <input type="button" value="Post"/>
Operation	<input type="button" value="Email"/> <input type="button" value="Sms"/> <input type="button" value="Push"/>
Receipients	<input type="button" value="app"/> <input type="button" value="user"/> <input type="button" value="listed"/>
CommTemplate	<input type="text"/>
Subject	<input type="text"/>
Target Details	<input type="text"/>
Action Delay	<input type="text" value="0"/>
<input type="button" value="Save"/>	

Figure 12: Information of An Action

- Firstly, the servers will process schema's information and transform these data into JSON object. Then, these JSON objects will be written to a JSON file. This file will be kept on the server.
- Next, whenever participants make a request to the server, these schemas will be used for extracting data and return to participants. The "controlId" of a request is used for determining which schema will be used.
- This request will trigger the RunTimeEnvironment (RTE) process on the server. The RTE will check if there is any data already existed in the server's cache. If the data is already in the cache, it will be used for retrieving data. Otherwise, the RTE will go forward and read the contents of the schema file.
- Then, the RTE will load the content of the JSON file such as a list of item, conditions, and actions, etc. The request's body will control what will the RTE return to the participants. The important point is that the returning data does not contain the whole information of the schema. For example: if a schema includes multiple question and conditions the RTE will try to get the first question and only return that question's data. When the participants the make same request, the RTE will automatically get the next question based on the pre-defined conditions of the schema.
- Finally, the app will be stopped when the last question is reached.

The schema is the most important and influential feature of MABIC application. Schemas hold all necessary information for displaying to participant's device. A schema works as an information trading central between the server and the client. Whenever the clients make a request, the servers will retrieve data from a schema and only select

required information. Then the RTE will convert it to JSON format and return to the client side. Overall, these processes will be automatically handled on the server side rather than client side.

2. Client Side

Mostly, the participants interact with the servers via a specific request. Clients send a request with JSON format, each request's content will be different depending on what kind of request is. When the server receives a request, it will process that request and return results back to clients. After receiving the result from servers, clients use that data to display to participant's screen. The workflow of the client can be described as in Figure 13.

2.1. Login

When opening an application for the first time, participants are required to enter their username and password. The login page contains two textboxes and a button that allowing participants to enter credential information, and then submitting that info to the server for authentication. If the username or the password is incorrect, the application will display an error message indicating the information is incorrect and require participants to input again. Otherwise, if both information is correct, a request contains username and password will be sent to the server. The format of body request is showed in Figure 14.

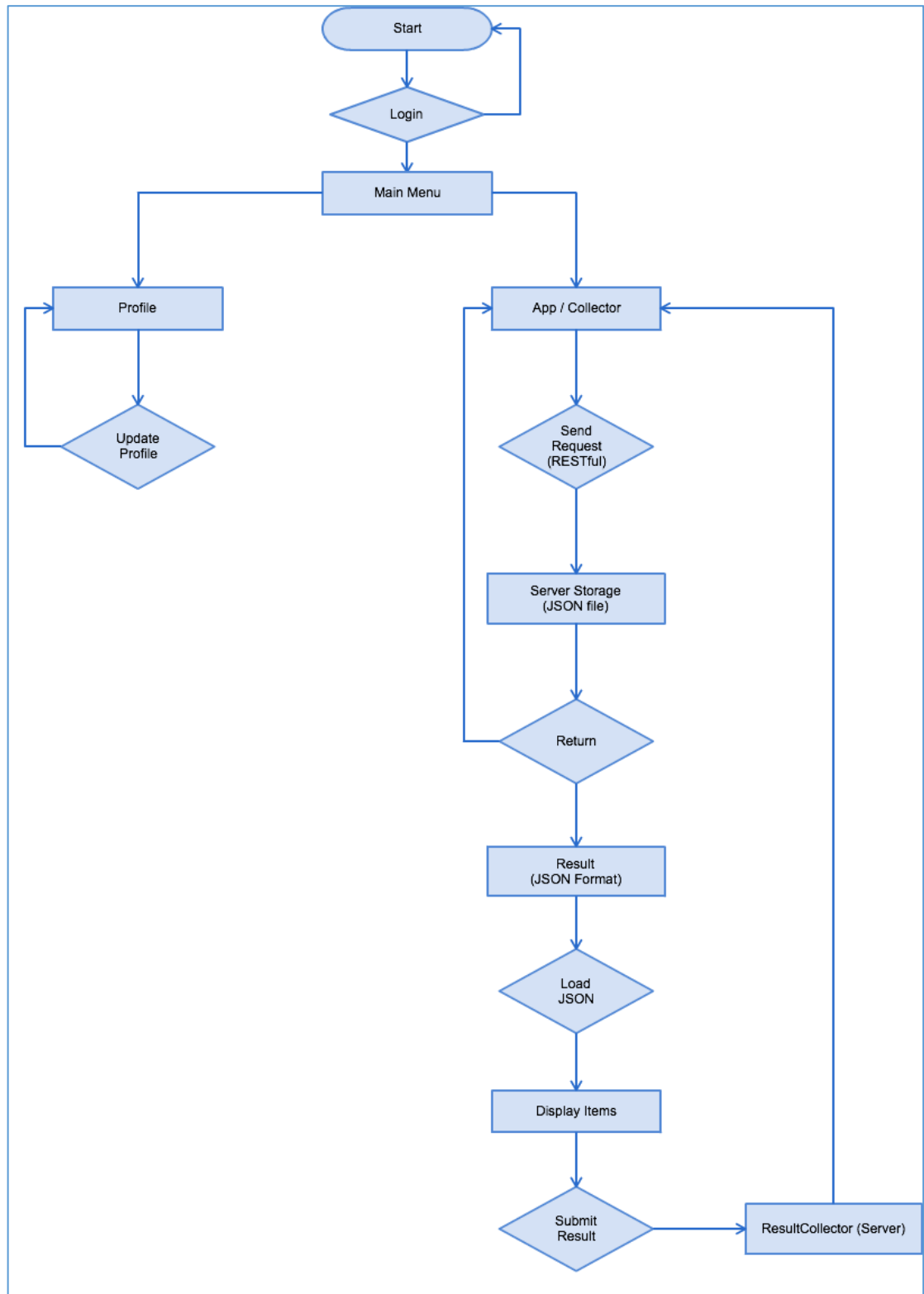


Figure 13: Client Workflow

Next, this request will be sent and processed by the server. After the username and password of the participant are checked as a valid user, the server will go forward and send data back to clients. This data has JSON format and contains numerous different information. However, there are only a few relevant fields that need to be a concern of such as reasoncode, token, and apps.

```
{"username": "part", "password": "ppp"}
```

Figure 14: An example of request body

a) Reasoncode

Reasoncode field is important information. This field indicates the status of return request from the server. If the code is 0, it means that username and password are valid. Therefore, the login process is successful and ready to use. However, if the reasoncode is returned 1, it means user's credential is incorrect. Thus, participants need to log in again. If the value of reasoncode is 2, this username is expired. In this case, this user must contact the administrator to resolve it. In the meantime, this user cannot login to the system.

b) Token

The token is important information. The best way is never to use the plain password to process a request, thus improve the security of the application. It will be unsafe; this password can be stolen while sending a request.

To prevent this problem, when the clients send a login request, it will encrypt the password and return a token. This token will be used as a temporary password for

processing the other request. This token is not permanent and only existing for a session. This token will be reset when participants log in to the application again.

c) **Apps**

The apps field is an array that including the number of apps and collectors belongs to the user. Each element in apps array is an app; each app contains default information such as appId, appName, appEmail, appPhone, parentApp and defaultApp.

- AppId, appName attributes show the id and name of the app. AppPhone is the phone number of the app owner.
- ParentApp indicates if the app is the main app or a data collector. If parentApp is an empty string, it will be the main app. Otherwise, it is a data collector, and parentApp value will be the appId of the main app.
- DefaultApp attribute shows if this app is a default app or not. If the value is false, then it is not a default app; otherwise, it is a default app. A default app will be automatically loaded after participants log in to the application.

Once this JSON data is obtained successfully from the server, it will be saved to the local storage so it can be used later.

2.2. App and Data Collector

There will be a sidebar menu that allows participants to select an app or a data collectors. When participants click on the main app via the app button, a request with appId, username, and token information will be sent to the server. All this information can easily to be retrieved from the local storage. Later, a response will be sent back to the client. Figure 15 is an example of an app's request response.


```

{
  "status": "OK",
  "token": "s0nyHp3nacGnIgYJ0QP5oUVs3gtCUh",
  "item": {
    "id": "choice1",
    "description": null,
    "items": null,
    "title": "How are you feeling today?",
    "instruction": null,
    "choices": [
      {
        "label": "1",
        "description": "Good",
        "value": "Good"
      },
      {
        "label": "2",
        "description": "Bad",
        "value": "Bad"
      }
    ],
    "imgs": null,
    "videos": null,
    "next": "f",
    "prev": "f",
    "type": "choice"
  },
  "nav": "false",
  "vars": null,
  "rollingStart": null,
  "frequency": null,
  "per": null,
  "responses": null,
  "respclients": null,
  "lastsync": null
}

```

Figure 15: A request result of an app

The result contains a few things: the status of the request, a current token of the participant, a question data and different properties of the app.

- The status of the request indicates if the request succeeds or not. If the status value is OK, it means this request is completed without any errors; otherwise, this request is incomplete.
- The item field shows the details of the question. It includes the id, description, titles and other attributes. This field will be different for each request. This item is retrieved through the schema on the server. However, only one question is returned for each request.
- Different attributes of the app are also returned. These attributes can be used to provide the proper behavior on the client side. For example: “nav” property indicates that if clients support the navigation between questions or not; the

“rollingStart” indicates if an app can be restarted when participants finish all questions or not; the “frequency” and “period” value supports to set up local notifications or reminders on the client side.

When clients send the first request to the server, either an app or data collector request, the response is similar. The response is used for displaying the question. This response will contain some essential details such as the id, type, title, and choices. However, the differences are the request body and the response of the next request.

Based on the kind of the question, the interface of that question will be designed differently. For example: if question type is a choice, the interface will contains multiple buttons. However, users can only pick one answer. If it is the check type, the interface will be similar to the choice type. But the participants can select multiple answers. For the range type, there will be a slider that allows participants to drag and move to change the value.

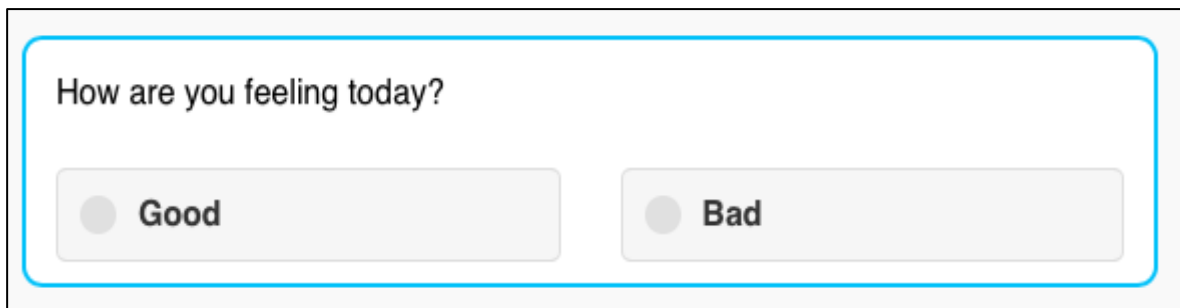


Figure 16: Example of a question

The “choices” property contains a list of the answers. Each answer has a label value displaying the order of the response. Description field indicates the content of the answer. This value will be sent back to the server later. Figure 16 display an example of a question interface. The size of choices array is the number of the answers to the question.

Also, “next” and “prev” attributes are used to specify if it is possible for participants to navigate back to the previous question or not.

In the main app, each request will only return one item from the server. Although there are many questions are created in the JSON file on the servers, there will be only one question returned. When the clients make a request, the servers will parse this request info and process it. Only the appID is needed for retrieving the next question. This approach simplifies the client’s request. So, the client’s developers do not need to concern about altering the request body to retrieve the other questions. Figure 17 is an example of client’s request to the server.

```
{
  "username": "part",
  "controlId": "59",
  "token": "4HQROZi9vFcps8SEzUkAuq97Cx3NaQ",
  "resp": [
    {
      "appId": "59",
      "name": "choice1",
      "value": "Bad",
      "time": 1476107548855
    }
  ]
}
```

Figure 17: An Example of Client's Request to Server

When participants submit an answer, a new request will be sent to the server. This request will also be in JSON format. This request is similar to the initial request. However, this request adds field name “rep”; this is the answer information of the question. This field contains appId; name field indicates the id of the question, the value was participant’s response and the time when the answer was selected. This data is kept in the local storage and will be dispatched to the server.

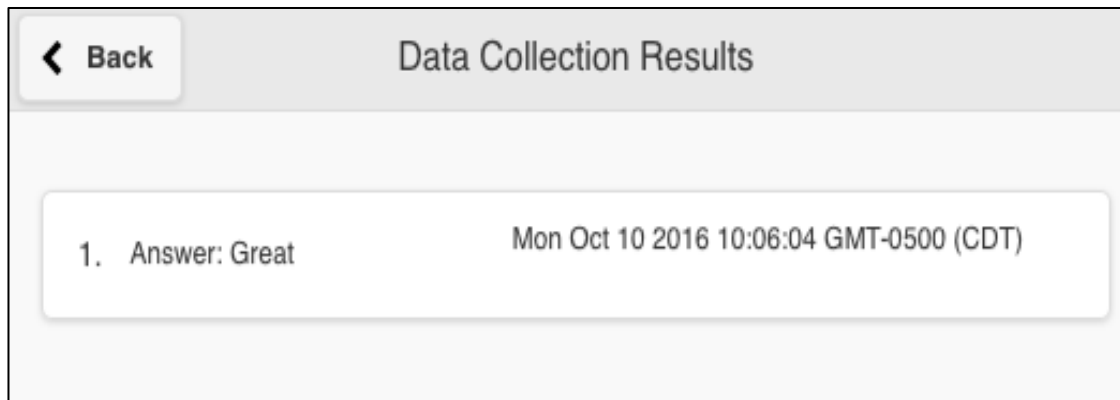


Figure 18: History View of Data Collector

For the data collectors, the behavior is similar to the main app. However, there are a few differences.

- Firstly, the responses can be different. The behaviors of the first request are the same. However, when clients send another request, it will not try to get a new question. Instead, it just sends participant's answer back to the server for saving purpose. A message will be displayed when the answer is successfully submitted.
- Another difference is the history button. There is a history view existed in a data collector app. This view is used for the read-only purpose. It shows all the submitted answers that participants already answered. The history data will be synchronized between clients and servers. Figure 18 shows an example of history view.

The workflow of the synchronization can be described as these following steps:

- Whenever the history view is accessed, the clients will make a request to the servers to retrieve the last timestamp info. This information indicates what is the last time that clients were synchronized with the server. There is also a

value named “lasttimesync” on the client. These values are being compared to determine which data on the server or the clients is updated.

- If the timestamp from the client side is newer or later than the server side, all new responses that were added after timestamp on the server side will be sent back and stored on the server. However, if the timestamp on the server is greater on the client side, it means that there are more updated data on the server side than the client. In this case, new data from the server will be retrieved and then merged with current data to the client. After this process, both server and client side data is updated.
- If the timestamp of clients and servers is equivalent, it does mean that the data does not need synchronization. This synchronization process makes sure that the data from the server and client will never be outdated.

2.3. Profile Update

From the menu on the client side, participants can easily access their profile information via the “Profile” button. Then, a profile page will be presented, and participants can look at their information including first name, last name, email, phone number, age and date of birth. This information is retrieved via sending a request to the server with username and token.

Figure 19: Result of profile’s request

Figure 19 shows a result of profile request. This response is also in JSON format; it includes:

- A status that indicates if the request is succeeded or not.
- It contains various details about the user account such as last name, first name, email and mobile phone.

- There is also a field named “additional.” This field is used for including the other attributes that belong to the user, e.g. age, date of birth. The reason for additional existed because the administrators on server side want to add more details to user information later. Instead of changing the entire structure of the user detail on the server side, it will be more convenient if the administrators only need to add more information to the “additional” field.

```
{
  "status": "OK",
  "lastName": "ABC",
  "firstName": "Huy",
  "userEmail": "huy.ng1147@gmail.com",
  "mobile": "2709919214",
  "additional": [
    {
      "displayName": "Age",
      "name": "age",
      "value": "50",
      "type": 2,
      "constraints": null,
      "enumVals": null,
      "length": 0
    },
    {
      "displayName": "Date of Birth",
      "name": "dob",
      "value": "10/29/1990",
      "type": 4,
      "constraints": "not empty",
      "enumVals": null,
      "length": 0
    }
  ]
}
```

Figure 19: Result of profile's request

Participants also can update their profile. The update can be achieved without a hitch by changing the content of text boxes that is associated with each field. For example: if participants want to change their last name, they can just go ahead and

modify the last name text box; or if they want to update their phone number, it can be accomplished by replacing new number on the phone number text box.

To get their changes updated, participants need to submit the new data to the server. A new request will be made. This request can be seen in Figure 20. When the server received the request, it will automatically update new value for the appropriate fields of the profile in the database. Then it will return the new values back to the clients. Thus, an updated profile information will be display immediately on participant's screen.

```
{
  "username": "part",
  "token": "715FqwINm78eUBgMFYMuReZiwwAg58",
  "firstName": "Huy",
  "lastName": "Nguyen",
  "userEmail": "huy.ng1147@gmail.com",
  "mobile": "2709919214",
  "additional": [
    {
      "name": "age",
      "value": "27"
    },
    {
      "name": "dob",
      "value": "05/20/1989"
    }
  ]
}
```

Figure 20: Update Profile Request

2.4. Local Notification

The fundamental purpose of the data collector is trying to collect data from participants frequently. It can be two times per day or one time a week, etc. Because of this reason, MABIC application supports local notification. The local notification acts as a reminder that prompts the participant's input data at the particular time of the day.

Participants can access local notification setting view via the “Setting.” A local notification setting view will be shown, as in Figure 21. The local notification is established based on the “frequency” and “period” value of data collector.

- If both values are not empty, then a reminder for that data collector will be automatically turned on.
- If the local notification status is on, for a specific time of the day or week, a reminder will display on the lock screen (if the device is locked). A banner notification will show up (if the device is unlocked) to inform participants that they must enter data value of those data collectors.
- The reminders setting will be loaded automatically when participant login to the application. However, participants can change this setting later by toggle the setting on or off.

All reminders can work offline without the internet. It does not send or receive any information from the server. These reminders only alter their behavior based on the setting of each app in the application.

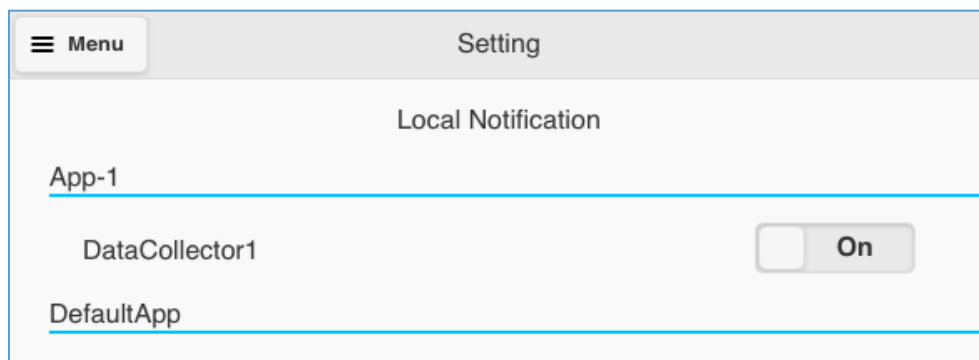


Figure 21: A Local Notification View

Chapter 6: Conclusion

The growing of the web services and mobile technology improves the modern communication. It allows the communication between the administrators and participants to be easier. But many existing systems are not good enough to fully support this kind of communication.

We have presented the architecture and how the workflow is working in the MABIC system. MABIC has the potential to be an excellent interactive communication system. It provides web services for the administrators and a mobile application for the participants. The content of the mobile app can be customized easily by the administrators. The server does not require the administrators has knowledge about programming skills; all the operation is simply done via a click or a drag-and-drop. The MABIC also support the instantaneous interaction by allow collecting data immediately from the participants or send the feedback to the participants. It has an automated mechanism which setting up the actions to send email, text message or push the notification to the client's device.

While this thesis has demonstrated the advantages and potentials of MABIC system, many features can be implemented for enhancing the communication between the servers and clients. The voice calls or video call can be integrated into the mobile application to enable the faster and direct communication. This improvement benefits both administrators and participants because it provides more information than before. Furthermore, video calls technology allows the administrators to stream the media content to multiple participants simultaneously and efficiently. This feature improves the

two-way communication by engaging more people to interactive with the system at the same time.

REFERENCES

- [1] Anghel, L. (2016, 3). *What Is PrimeFaces?* Retrieved from Developer.com:
<http://www.developer.com/java/data/what-is-primefaces.html>
- [2] Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., . . . Wallnau, K. (2000). *Volume II: Technical concepts of component-based software engineering*. Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute.
- [3] Chapanis, A., Ochsman, R. B., Parrish, R. N., & Weeks, G. D. (1972). Studies in interactive communication: I. The effects of four communication modes on the behavior of teams during cooperative problem-solving. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 14(6), 487-509.
- [4] Dikanski, A., Steinegger, R., & Abeck, S. (2012). Identification and Implementation of Authentication and Authorization Patterns in the Spring Security Framework. *Procs. 6th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*. IARIA, Rome, Italy, 14-20.
- [5] Eng, T. R., Gustafson, D. H., Henderson, J., Jimison, H., & Patric, K. (1999). Introduction to evaluation of interactive health communication applications 1. *American journal of preventive medicine*, 16(1), 10-15.
- [6] *Expression-Based Access Control*. (2016). Retrieved from Spring by Pivotal:
<http://docs.spring.io/spring-security/site/docs/current/reference/html/el-access.html>

- [7] Fulk, J., Flanagan, A. J., Kalman, M. E., Monge, P. R., & Ryan, T. (1996). Connective and communal public goods in interactive communication systems. *Communication Theory*, 6(1), 60-87.
- [8] Georgakopoulos, D., Hornick, M., & Sheth, A. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2), 119-153. doi:10.1007/BF01277643
- [9] Han, J. (1999). An approach to software component specification. *Proceedings of International Workshop on Component-Based Software Engineering*.
- [10] Harris, P. A., Taylor, R., Thielke, R., Payne, J., Gonzalez, N., & Conde, J. G. (2009). Research Electronic Data Capture (REDCap) - A metadata-driven methodology and workflow process for providing translational research informatics support. *Journal of Biomedical Informatics*, 42(2), 377-381. doi:10.1016/j.jbi.2008.08.010
- [11] Huser, V., Rasmussen, L. V., Oberg, R., & Starren, J. B. (2011). Implementation of workflow engine technology to deliver basic clinical decision support functionality. *BMC Medical Research Methodology*, 11(1), 1. doi:10.1186/1471-2288-11-43
- [12] Jablonski, S., & Bussler, C. (1996). Workflow management: modeling concepts, architecture and implementation.
- [13] Jang, J., Choi, Y., & Zhao, L. J. (2004). An Extensible Workflow Architecture through Web Services. *International Journal of Web Services Research*, 1(2), 1-15.

- [14] Kozaczynski, W., & Booch, G. (1998). Component-based software engineering. *IEEE Software*, 15(5), 34-36. doi:<http://dx.doi.org/10.1109/MS.1998.714621>
- [15] Laymann, F., & Roller, D. (1997). Workflow-based Applications. *IBM Systems Journal*, 36(1), 102-123. doi:10.1147/sj.361.0102
- [16] Leavens, G. T., & Sitaraman, M. (2000). *Foundations of Component-based Systems*. New York, NY: Cambridge University Press.
- [17] Litchfield, R. E., Oakland, M. J., & Anderson, J. A. (2000). Improving dietetics education with interactive communication technology. *Journal of the American Dietetic Association*, 100(10), 1191-1194.
- [18] Logic, B. (2016). *Branch Logic*. Retrieved from Qualtrics Support: <https://www.qualtrics.com/support/survey-platform/survey-module/survey-flow/standard-elements/branch-logic/>
- [19] Miyasaka, K., Suzuki, Y., Sakai, H., & Kondo, Y. (1997). Interactive communication in high-technology home care: videophones for pediatric ventilatory care. *Pediatrics*, 99(1), e1-e1.
- [20] Murray, E., Burns, J. S., See, T. S., Lai, R., & Nazareth, I. (2005). Interactive Health Communication Applications for people with chronic disease. *Cochrane Database Syst Rev*, 4.
- [21] Nyanchama, M., & Osborn, L. S. (1994). Access Rights Administration in Role-Based Security Systems. *DBSec*, 37-56.
- [22] Oliveira Valente, M. d., Tirelo, F., Leao, D. C., & Silva, R. P. (2005). An Aspect-Oriented Communication Middleware System. In M. T. Oliveira Valente, F. Tirelo, D. C. Leao, & R. P. Silva, *On the Move to Meaningful Internet Systems*

2005: *CoopIS, DOA, and ODBASE* (pp. 1115-1132). Springer Berlin Heidelberg.
doi:10.1007/11575801_12

- [23] Prestwich, S., & Mudambi, S. (1995). Improved branch and bound in constraint logic programming. *Lecture Notes in Computer Science*, 976, 533-548.
doi:10.1007/3-540-60299-2_32
- [24] Schalk, C. (2005, April). *Introduction to JavaServer Faces - What is JSF?*
Retrieved from Oracle: <http://www.oracle.com/technetwork/topics/index-090910.html>
- [25] Sirbi, K., & Kulkarni, P. J. (2010). Stronger Enforcement of Security Using AOP. *Journal Of Computing*, 2(6), 99-105.
- [26] Stohr, E. A., & Zhao, J. L. (2001). Workflow Automation: Overview and Research Issues. *Information Systems Frontiers*, 3(3), 281-296.
doi:10.1023/A:1011457324641
- [27] Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., . . . Goble, C. (2013). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(Web Server issue), W557–W561. doi:10.1093/nar/gkt328