International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-2, Issue-3, July 2012

Mobile App Design Tool for Smartphones: A Tutorial

Hak. J. Kim, Jonathan Modell

Abstract*: *The paper presents the basics of mobile application creation for smartphones using the visual programming tool, 'App Inventor for Android (AIA)'. The AIA was developed by Google to enable non-programmers to easily build mobile applications by dragging and dropping the block-based interfaces, like blocks of Lego, instead of writing lines and lines of code. It allows users to immediately build applications (e.g., location services and games) that interface with mobile technologies.

Index Terms*: *App Inventor, Mobile App, Smartphones, Designer, Block Editor.

I. INTRODUCTION

With the popularity of smartphones, mobile applications are increasingly gaining interest to younger generations as well as students who are interested in computer science [1]. *App Inventor for Android* [2, 3, 4] has great potential because individuals without any prior programming skills can easily build their own applications with real-world impact. Students can become excited to learn by tinkering with their most beloved devices (e.g., smartphones)

The main objective of this project is to build the smart parking management system to solve current parking problems on the university campus. It aims to provide real-time parking information as well as support a more sustainable campus by reducing the amount of carbon dioxide. In this paper, we will introduce the mobile app design tool, AIA, and then attempt to design a mobile application for campus parking management.

II. WHAT IS APP INVENTOR

App Inventor [3] is a mobile applications design tool consisted of two major parts; Component Designer and Block Editor. It allow users without prior programming experience to develop mobile applications. App Inventor [4] lets users develop applications for Android phones using a web browser and either a connected phone/tablet or an emulator .

Figure 1 shows the architecture of App Inventor. The App Inventor Server stores a developer's work and help him/her keep track of a project. The *App Inventor Designer* [5] is mainly used to create the interface of a user's own apps and put them together from its functional components, such as Basic, Media, Animation, Social, and Sensor. It is implemented as a web application, so you load it just like a normal website into your browser by entering the appropriate web address. After finishing the design of applications, the *App Inventor Block Editor* should be used to specify how the components of *App Inventor Designer* [3] should behave. The more detail process will xplained next session.

Instead of the traditional method of debugging, your app appears on the phone as you add pieces to it, so you can test your work as you build. When you're done, you can package your app and produce a stand-alone application to install. If you don't have an Android phone, you can build your apps using the *Android emulator* [4], software that runs on your computer and behaves (with a few exceptions) just like the phone.

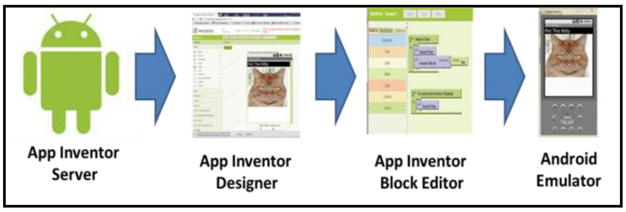

Figure 1: the Architecture of App Inventor

III. DEVELOPMENT ENVIRONEMENT

The App Inventor development environment [5] is supported for Mac OS X, GNU/Linux, and Windows operating systems, and several popular Android phone models. Applications created with App Inventor can be installed on any Android phone.

A. Objects

An object represents an instance of a class such as Form, Control, or Component. An everyday object such as an automobile also has properties, methods, and events.

B. Property

A *property* is an attribute of an object that defines one of the object's characteristics. Examples of properties are the size, color, or screen location of the object. Other properties could include aspects of the object's behavior, such as whether it is enabled or visible.

C. Method

A *method* is an action that an object can perform. For example, Add is a method of the ListPicker object, because it adds a new entry into the list.

D. Event

An event is an action recognized by an object, such as clicking the mouse or pressing a key, and for which you can write code to respond. Events can occur as a result of a user action or program code, or they can be triggered by the system.

Hak Ju Kim, Department of Information Technology and Quantitative Methods, Hofstra University, Hempstead, NY 11549, USA, 1.516.463-4529..
Jonathan Modell, Department of Information Technology and Quantitative Methods, Hofstra University, Hempstead, NY 11549, USA, 1.516.463-4529.,.

E. Example

Figure 2 shows an example of automobile. An **object** (Autombile) has **properties** (Color, Length, and Height), responds to **events** (Collide with object), and can perform **methods** (Drive). An automobile also has known methods or actions that it can perform. For example, a 'Fill her up' method (filling it with gasoline). A 'drive' method (expelling its contents such as gasoline, oil, winshield wiper fluid). An automobile's properties include visible attributes such as its length, model, and color. Other properties describe its state (running, idle, off), or attributes that are not visible, such as its age.

Figure 2: Objects, Properties, Events, and Methods of Automobile

IV. COMPONENTS OF APP INVENTOR

As addressed in the above, the App Inventor consists of two components - the '*Component Designer*' for specifying the visual *components* of an application and the '*Blocks Editor*' for creating *behaviors* for the *components* [9].

1. Component Designer

The *Component Designer* provides the components to design your application, such as Basic, Media, Animation, Social, Sensor, Arrangement, and Other. Each component can have methods, events, and properties. Figure 3 shows a screen shot of *App Inventor Designer*.

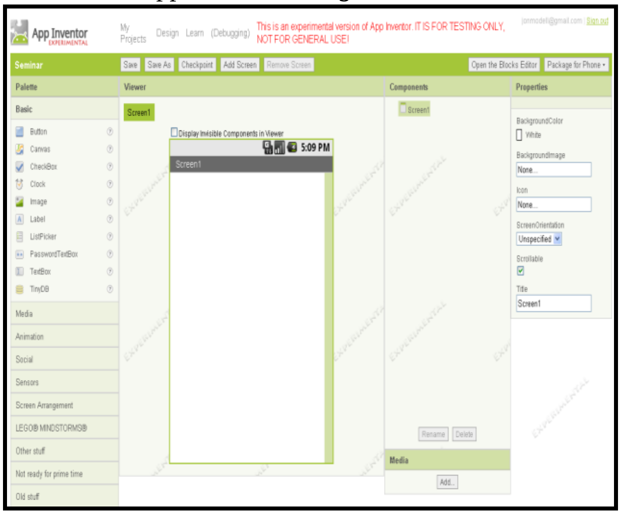

Figure 3: App Inventor Designer

A. Basic Component

Figure 4 shows the list of *Basic Component*. Each component is briefly described as follows:

- **Button** is a component that users touch to perform some action in your application.
- **Canvas** is a two-dimensional rectangular panel on which drawing can be done and sprites can be moved.
- **CheckBox** can detect user taps and can change their Boolean state in response.
- **Clock** is used to create a timer that signals events at regular intervals.
- **Image** is to represent images that users select and manipulate.
- **Label** is a component used to show text.
- **ListPicker** is that users tap a list picker component to select one item from a list of text strings.
- **PasswordTextBox** is that users enter passwords in a password text box component, which hides the text that has been typed in it.
- **TextBox** is to enter text in a text box.
- **TinyDB** is to store data that will be available each time the app runs.

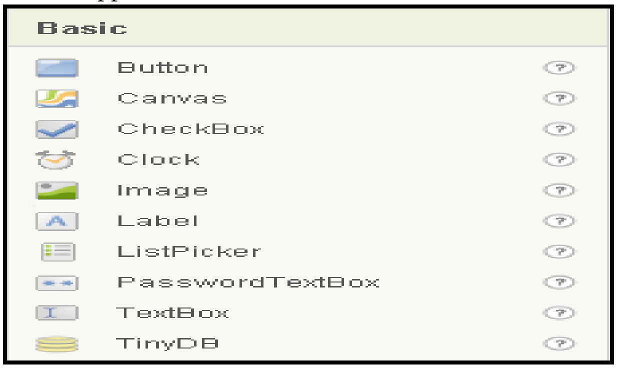

Figure 4: Basic Component

B. Media Component

Figure 5 shows the list of *Media Component*. Each component is briefly described as follows:

- **Camera** is used to take a picture on the phone.
- **ImagePicker** is used to choose an image from your image gallery.
- **Player** is used to play an audio or video file, or to vibrate the phone.
- **Sound** is used to play an audio file, or to vibrate the phone.
- **VideoPlayer** is used to play a video file.

Figure 5: Media Component

C. Animation Component

Figure 6 shows the list of *Animation Component*. Each component is briefly described as follows:

- **Ball** is a particular kind of sprite (animated object) that looks like a ball.
- **Image Sprite** is an animated object that can interact with a canvas, balls, and other image sprites.

Figure 6: Animation Component

D. Social Component

Figure 7 shows the list of *Social Component*. Each component is briefly described as follows:

- **ContactPicker** is used to let the user choose an entry from the Android contact list.
- **EmailPicker** is used to let the user enter a user's email address from the Android contact list.
- **PhoneCall** is used to dial the phone and make a call.
- **PhoneNumberPicker** is used to allow users to choose a phone number from a list of Android contacts' phone numbers.
- **Texting** is used to allow users to send and receive text messages.
- **Twitter** is a non-visible component to enable communication with Twitter.

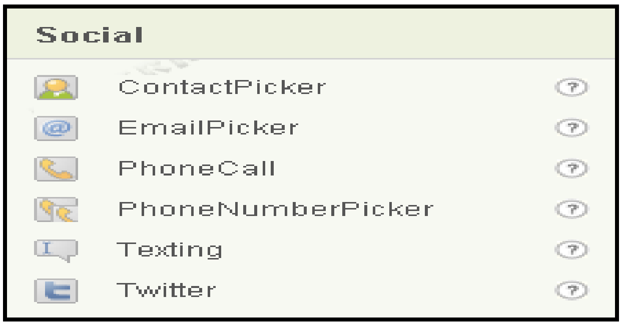

Figure 7: Social Component

E. Sensors Component

Figure 8 shows the list of Sensors *Component*. Each component is briefly described as follows:

- **AccelerometerSensor** senses the Android device's accelerometer, which detects shaking and measures acceleration in three dimensions.
- **LocationSensor** provides the Android device's location, using GPS if available and an alternative method otherwise, such as cellular towers or known wireless networks.
- **OrientationSensor** uses an orientation sensor component to determine the phone's spatial orientation.

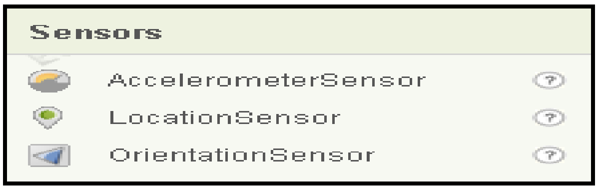

Figure 8: Sensors Component

F. Screen Arrangement Component

Figure 9 shows the list of *Screen Arrangement Component*. Each component is briefly described as follows:

- **HorizontalArrangement** uses a horizontal arrangement component to display a group of components laid out from left to right.
- **TableArrangement** uses a table arrangement component to display a group of components in a tabular fashion.
- **VerticalArrangement** uses a vertical arrangement component to display a group of components laid out from top to bottom, left-aligned.

Figure 9: Screen Arrangement Component

G. Other Stuff

Figure 10 shows the list of *Other Stuff*. Each component is briefly described as follows:

- **ActivityStarter** launches another activity from your application.
- **BarcodeScanner** reads a 1-dimensional barcode or 2-dimensional barcode (QR code). In order for this component to work, the Barcode scanner application from ZXing must be installed on the phone.
- **Notifier** uses a notifier to display notices and alerts to users of your app, and also to log information that can help you debug your application.
- **SpeechRecognizer** listens to the user speaking and convert the spoken sound into text using Android's speech recognition feature.
- **TextToSpeech** uses a text-to-speech component to have the device speak text audibly.
- **TinyWebDB** is a non-visible component that communicates with a Web service to store and retrieve information.
- **Web** is a non-visible component that provides functions for HTTP GET and POST requests.

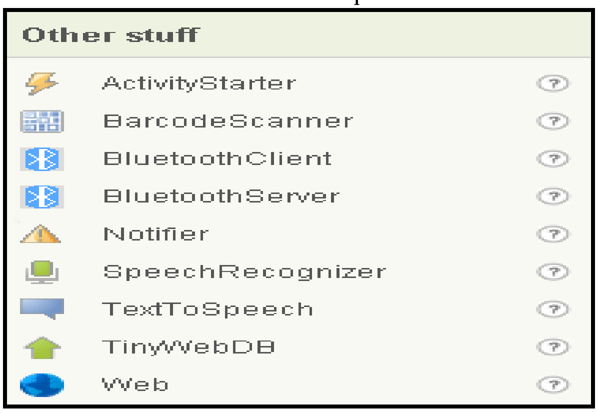

Figure 10: Other Stuff Component

2. Block Editor

The *Blocks Editor* is used to assign behavior to the components. It is where you indicate what events should cause action in the app and what the resulting algorithm is after the event occurs. It allows a user to set of different properties. It can make a reference to a component via a variable/argument and manipulate dynamically. Figure 11 shows a screen shot of *Block Editor*.

The *Block Editor*, where you assemble program blocks, specifies how the components should behave. You assemble programs visually, fitting pieces together like pieces of a puzzle. There are seven block categories, not including the *My Blocks* section.

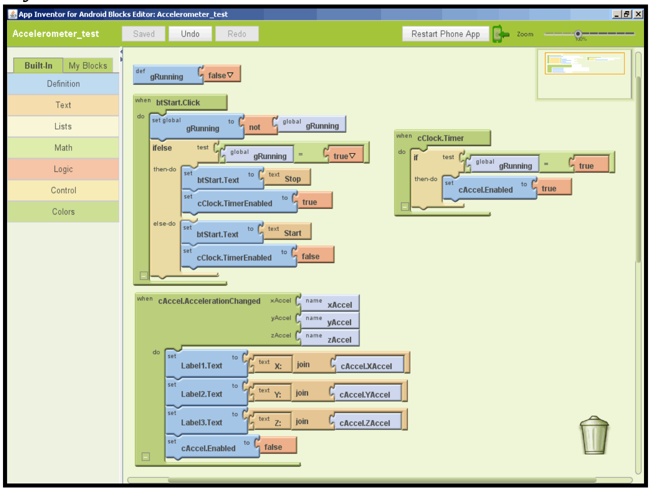

Figure 11: Block Editor

A. *Definition Blocks*

There are four *Definition Blocks*: procedure, procedureWithResult, name, and variable, as shown Figure 12. These blocks allow user to automatically generate blocks definitions related to *App Inventor* components.

procedure	Collects a sequence of blocks together into a group	to procedureName arg do
procedureWithResult	Same as a procedure block, but calling the procedure returns a result. After the procedure executes, the result is returned to the block connected to the return socket.	to procedureWithResult arg do return
name	Creates a named argument you can use when calling a procedure.	name name
variable	Creates a value that can be changed while an app is running, and gives that value a name.	def variable

Figure 12: Definition Blocks

B. Text Blocks

Figure 13 shows the list of *Text Blocks* and each component is briefly described. These blocks create, manipulate, and destroy text. Text is usually used as a user interface. Currently there are 20 text block types.

text	Contains A Text String	text text
equals	Tests whether two given values are equal	=
join	Appends the second given string to the first	join
make text	Joins all given values into one text string	call make text text
length	Returns the length of the given string	call length text
text<	Reports whether the first text string argument is alphabetically less than the second text string	text< text1 text2
text=	Reports whether the text strings are identical, i.e., have the same characters in the same order	text= text1 text2
text>	Reports whether the first text string argument is alphabetically greater than the second text string	text> text1 text2
upcase	Returns a copy of its text string argument converted to uppercase	upcase text
downcase	Returns a copy of its text string argument converted to lowercase	downcase text
trim	Returns a copy of its text string argument with any leading or trailing spaces removed	trim text
starts at	Returns the character position where the first character of piece first appears in text, or 0 if not present	call starts at text piece
contains	Returns true if piece appears in text; otherwise, returns false	call contains text piece
split at first	Divides the given text into two pieces using the location of the first occurrence of at as the dividing point	call split at first text at
split at first of any	Divides the given text into a two-item list, using the location of any item in the list at as the dividing point	call split at first of any text at
split	Divides text into pieces using at as the dividing points and produces a list of the results	call split text at
split at any	Divides the given text into a list, using any of the items in at as the dividing point, and returns a list of the results	call split at any text at
split at spaces	Divides the given text at any occurrence of a space, producing a list of the pieces	call split at spaces text
segment	Extracts part of the text starting at start position and continuing for length characters	call segment text start length
replace all	Returns a new text string obtained by replacing all occurrences of the substring with the replacement	call replace all text segment replacement

Figure 13: Text Blocks

C. List Blocks

List blocks are used for creating and manipulating lists which can be combined with the for each block to do a set series of operations on every value in the list. Figure 14 shows the list of *List Blocks* and each component is briefly described. List is a necessity in almost every applications regardless of what programming language is used. This is the easiest way to create and manipulate a set of values, items, and elements in an ordered fashion.

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication

make a list	Creates a list from the given blocks. If you don't supply any arguments, this creates an empty list, which you can add elements to later	call make a list item
select list item	Selects the item at the given index in the given list. The first list item is at index 1	call select list item list index
replace list item	Inserts replacement into the given list at position index. The previous item at that position is removed	call replace list item list index replacement
remove list item	Removes the item at the specified position from the list	call remove list item list index
insert list item	Inserts an item into a list at the specified position	call insert list item list index item
length of list	Returns the number of items in the list	call length of list list
append to list	Adds the items in the second list to the end of the first list	call append to list list1 list2
add items to list	Adds the given items to the end of the list.	call add items to list list item
is in list?	If thing is one of the elements of the list, returns true; otherwise, returns false	call is in list? thing list
position in list	Returns the position of thing in the list, or 0 if it's not in the list	call position in list thing list
pick random item	Picks an item at random from the list	call pick random item list
is list empty?	If list has no items, returns true; otherwise, returns false	call is list empty? list
copy list	Makes a copy of a list, including copying all sublists	call copy list list
is a list?	If thing is a list, returns true; otherwise, returns false	call is a list? thing
list to csv row	Interprets the list as a row of a table and returns a CSV (comma-separated value) text representing the row	call list to csv row list
list to csv table	Interprets the list as a table in row-major format and returns a CSV text representing the table	call list to csv table list
list from csv row	Parses a text as a CSV (comma-separated value) formatted row to produce a list of fields	call list from csv row text
list from csv table	Parses a text as a CSV formatted table to produce a list of rows, each of which is a list of fields	call list from csv table text

Figure 14: List Blocks

D. Logic Blocks

Figure 16 shows the list of *Logic Blocks* and each component is briefly described.

True	This block represents the constant value true. Use it for setting Boolean property values of components, or as the value of a variable that represents a condition	true
False	This block represents the constant value false	false
Not	This block performs logical negation, returning false if the input is true, and true if the input is false	not
Equals	This block tests whether its arguments are equal.	=
And	This block tests whether all of a set of logical conditions are true. The result is true if and only if all the tested conditions are true	and test
Or	This block tests whether any of a set of logical conditions are true. The result is true if one or more of the tested conditions are true	or test

Figure 16: Logic Blocks

E. Control Blocks

Figure 17 shows the list of *Control Blocks* and each component is briefly described.

If	Tests a given condition. If the condition is true, performs the actions in a given sequence of blocks; otherwise, the blocks are ignored	if test then-do
ifelse	Tests a given condition. If the result is true, performs the actions in the **then-do** sequence of blocks; otherwise, performs the actions in the **else-do** sequence of blocks.	ifelse test then-do else-do
choose	Tests a given condition. If the condition is true, performs the actions in the **then-do** sequence and returns the **then-return** value; otherwise, performs the actions in the **else-do** sequence of blocks and returns the **else-return** value.	choose test then-do then-return else-do else-return
foreach	Runs the blocks in the do section for each item in the list in list. Use the given variable name to refer to the current list item.	foreach item in list do
for range	Runs the block in the do section for each numeric value in the range from start to end, stepping the value each time. Use the given variable name to refer to the current value.	for range variable start end step do
while	Tests the **test** condition. If true, performs the action given in **do**, then tests again. When **test** is false, the block ends.	while test do
get start text	Returns the text passed to this app when the app was started, if any.	call get start text
close screen	Closes the app	call close screen
close screen with result	Closes the app and sets the variable APP_INVENTOR_RESULT to the given value	call close-screen-with-result result
close application	Closes the current application	call close application
Open another screen	Opens another screen. Takes the screen name as the argument	call open another screen screenName
Open another screen with start text	Opens another screen and passes the start text to it. Takes the screen name and the start text to pass as the arguments	call open another screen with start text screenName startText

Figure 17: Control Blocks

F. Math Blocks

The math blocks handle mathematical functions suitable for calculator applications. Figure 18 shows the list of *Math Blocks* and each component is briefly described.

number	Specifies a numeric value	
>	Compares two given numbers. If the first is larger, returns true; otherwise, returns false	
<	Compares two given numbers. If the first is smaller, returns true; otherwise, returns false	
≤	Compares two given numbers. If the first is smaller than or equal to the second, returns true; otherwise, returns false	
≥	Compares two given numbers. If the first is greater than or equal to the second, returns true; otherwise, returns false	
=	Tests whether two given values are equal. If so, returns true; otherwise, returns false	
+	Returns the sum of two given numbers	
-	Returns the result of subtracting the second number from the first	
×	Returns the product of two given numbers	
/	Returns the result of dividing the first number by the second	
sqrt	Returns the square root of the given number.	
random fraction	Returns a random value between 0 and 1	
random integer	Returns a random integer value between the given values, inclusive.	
random set seed	Use this block to generate repeatable sequences of random numbers	
negate	Returns the negative of the given number	
Min	Returns the smallest of a given set of numbers	
Max	Returns the largest of a given set of numbers	
quotient	Returns the result of dividing the first number by the second and discarding any fractional part of the result	
remainder	Remainder(a,b) returns the result of dividing a by b and taking the remainder	
modulo		
abs	Returns the absolute value of the given number	
round	Rounds the given number to the nearest integer and returns the result	
floor	Calculates the greatest integer that's less than or equal to the given number	
ceiling	Returns the smallest integer that's greater than or equal to the given number	
expt	Raises the first given number to the power of the second and returns the result	
exp	Returns *e* (2.71828...) raised to the power of the given number and returns the result	
log	Returns the natural logarithm of the given number	
sin	Returns the sine of the given number in degrees	
cos	Returns the cosine of the given number in degrees	
tan	Returns the tangent of the given number in degrees	
asin	Returns the arcsine of the given number in degrees	
acos	Returns the arccosine of the given number in degrees	
atan	Returns the arctangent of the given number in degrees	
atan2	Returns the arctangent of y/x, given y and x	
format as decimal	Format a number as a decimal with a given number of places after the decimal point.	
is a number?	Returns true if the given object is a number, and false otherwise	

Figure 18: Math Blocks

G. Color Blocks

The *Color Blocks* contain several blocks corresponding to commonly used colors. Figure 19 shows the list of *Color Blocks* and each component is briefly described. These blocks have two types one is the text color setting and the other is the background color setting. These same colors can also be selected from the color dropdown lists in the Designer's properties panel.

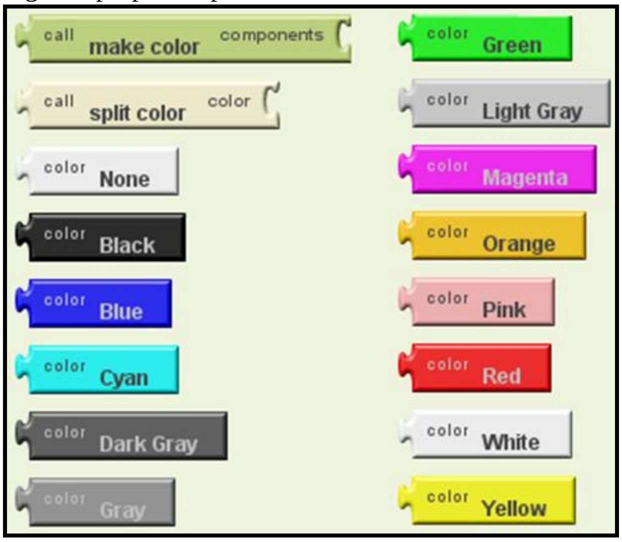

Figure 19: Color Blocks

V. AN EXAMPLE: DESIGNING CAMPUS PARKING SYSTEM

In this project, we designed a test-bed for performance evaluation of the RFID-enabled and mobile app-based parking management system in a controlled and repeatable laboratory environment. The test-bed consists of a real-time RF channel simulator, Wi-Fi 802.11 access points, RFID tags, and a laptop loaded with the positioning algorithm and its associated user interface (i.e., mobile app) [6].

The parking management platform consists of four systems [7]; RFID system, Wi-Fi systems, database management system (DBMS), and mobile systems loaded with our mobile app. The web-based mobile app will use the University Wi-Fi network to triangulate the position of the user's smart phone. It will provide a view of the University's parking lot layout and the user's location on the smart phone (combining the GPS guidance system and the RFID technology).

The mobile app will provide real-time parking information (e.g., available parking spaces) and also guide the user giving them turn by turn directions to the destination.

Using the mobile app for android, the preliminary test-bed for integrating the RFID system, wireless network, and mobile devices. The methodology for this test-bed will be a lab-based approach (experiment). This pilot study will be carried out in the University's computer networking lab and use some facilities (power source, database server, wireless access points, and PCs).

The additional equipment for a test-bed includes RFID Tags, RFID Readers, and RFID software. Figure 20 shows web-based mobile app prototype which provides a map with parking information like available parking lots and spaces.

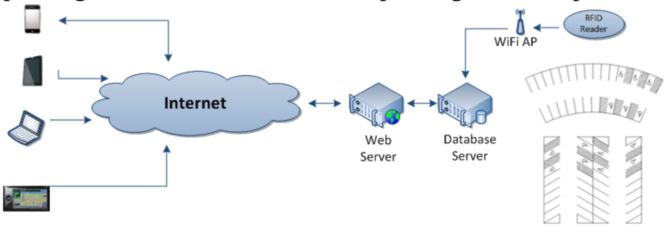

Figure 20: A Test-Bed Architecture

VI. CONCLUDING REMARKS

This paper introduces the *App Inventor for Android* as a tool to develop mobile applications, including its definitions, components, and architecture. The benefit of this software is that there is no prior experience in programming, but improvement in the knowledge of computer programming.

Using this mobile app design tool, we plan to build the campus parking management system for solving current parking problems in university campus by providing the real-time parking information and also for supporting more sustainable campus by reducing the amount of carbon dioxide. This system also will be applied to many fields, such as central business districts, airports, transit stations, and shopping centers.

REFERENCES

1. Cook, D., and Das, S. (2012) Pervasive Computing at Scale: Transforming the State of the Art, Pervasive and Mobile Computing, vol. 8, issue 1, pp. 22-35.
2. MIT, "App Inventor for Android", http://appinventoredu.mit.edu/ (accessed February 20, 2012).
3. Tyler, J. (2012) App Inventor for Android: Build Your Own Apps-No Experience Required !, Wiley Inc.
4. Wolber, D., Abelson, H., Spertus, E., and Looney, L. (2011) App Inventor: Create Your Own Android Apps, O'Reilly.
5. Kloss, J. (2012) Android Apps with App Inventor, Pearson Education, Inc.
6. Castro, L. & Fosso Wamba, S. (2007). An inside look at RFID technology. Journal of Technology Management & Innovation, 2(1), pp. 128-140.
7. Roussos G. and Kostakos V. (2009) RFID in Pervasive Computing: State-of-the-art and Outlook, Pervasive and Mobile Computing, Vol. 5(1), pp.110-131.

AUTHORS PROFILE

Hak J. Kim is an Associate Professor of Information Technology and Quantitative Methods at the Hofstra University. He is a Director of Computer and Network Lab in the Zarb School of Business. His current research interests include mobile computing & ubiquitous business, social networking services & social media, and cyber space & cyber security. In this workshop, he is interested in applying mobile computing technologies to smartphones and analyzing how his research interests are incorporated in the real-world businesses.

Jonathan Modell is currently an MBA graduate student at the Zarb School Of Business/Hofstra University with a focus on Information Technology. With over fifteen years of industry experience, he currently owns and operates a small computer consulting firm focusing on the needs of smaller emerging businesses on Long Island.