

# Make This

## Chapter 11 - Using TinyWebDB

TinyWebDB is an App Inventor component designed to store data persistently in a database located on the web. Web storage has an advantage over data storage on one particular device (such as with TinyDB, which you used in Chapter 9). With web storage, multiple devices can share and use the data stored. For instance, if you distributed a game app, you would use web storage for the high score files. In this exercise, you'll use the **TinyWebDB** component to store data and retrieve it from a database that is deployed on the web while building a shopping list app that multiple people can access.

This tutorial teaches the following skills:

- **Using the TinyWebDB Component**

**Note:** Before attempting this exercise, complete the Chapter 2 and 3 exercises to familiarize yourself with the App Inventor interface, getting your Android device connected, and starting a new project.

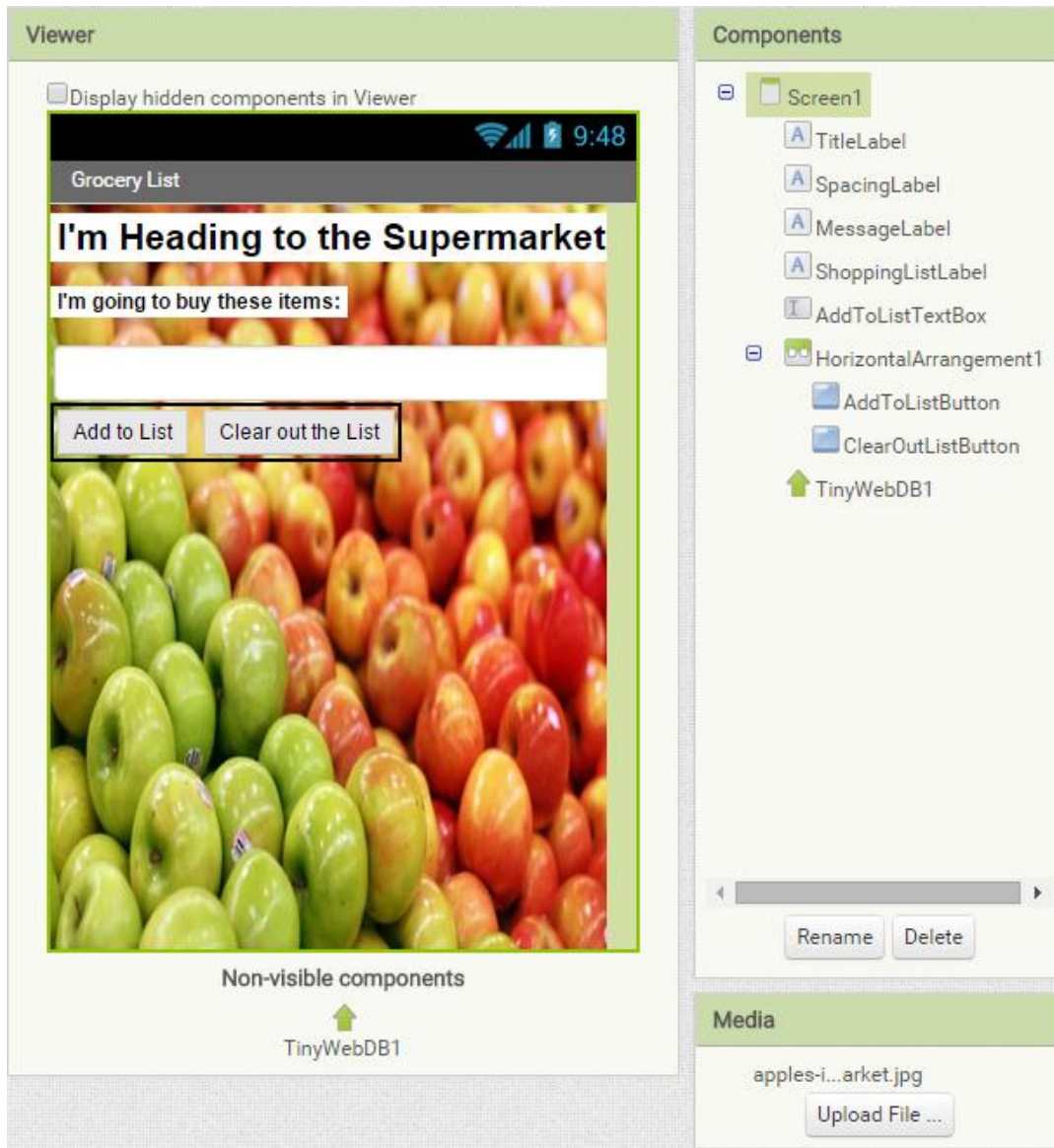
## Building the Shopping List App

1. Navigate to <http://appinventor.mit.edu/explore/>. If necessary, sign in with your Google Account.
2. Start a new project named *ShoppingListApp*. Change the **Title** of **Screen1** to *Grocery List*. Set the **BackgroundImage** to *apples-in-the-supermarket.jpg* (file provided...upload it to App Inventor first). Set the **BackgroundColor** to *None*.
3. Connect App Inventor to your Android device.

### Part 1 - Constructing the Interface

The interface allows the user to view items already on the grocery list (if any) and add their own items to the list.

Your layout for your interface should look like this:



Build the interface shown above by dragging out the components shown in the following table:

Component type	Palette Group	What you'll name it	Purpose
Label	User Interface	TitleLabel	Displays title of app
Label	User Interface	SpacingLabel	Label to add space between TitleLabel and MessageLabel
Label	User Interface	MessageLabel	Displays message below title
Label	User Interface	ShoppingListLabel	Displays items on the shopping list
TextBox	User Interface	AddToListTextBox	Allows user to enter items to be added to the shopping list

HorizontalArrangement	Layout	HorizontalArrangement1	Area to line up the AddToListButton and ClearOutListButton
Button	User Interface	AddToListButton	Button add item entered to the list
Button	User Interface	ClearOutListButton	Button to clear all items from the current list
TinyWebDB	Storage	TinyWebDB1	Non-visible component that that communicates with a Web service to store and retrieve information

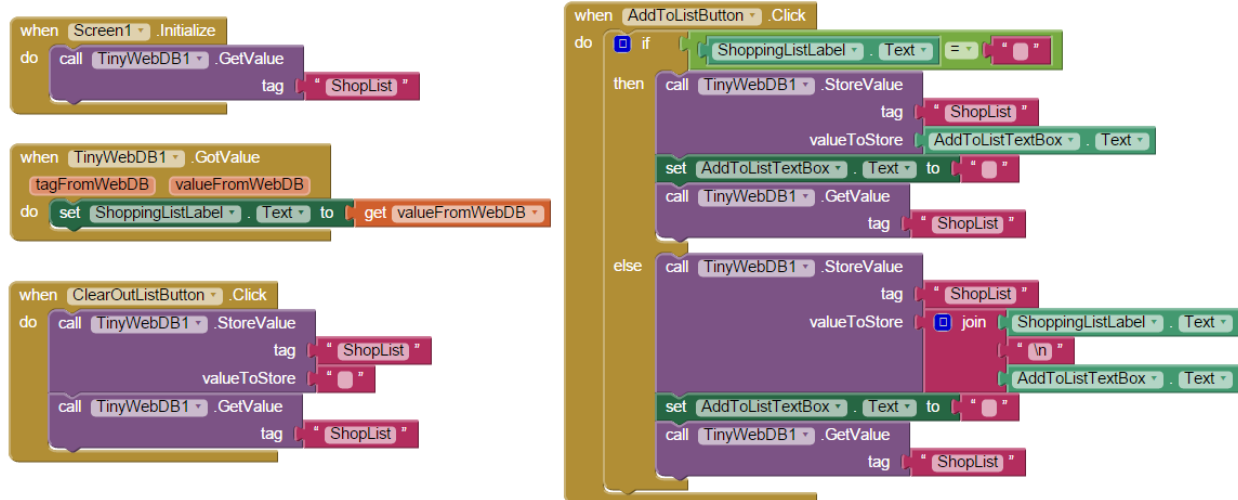
Set the properties of each component in the following ways:

- Change the **Text** for **TitleLabel** to *I'm Heading to the Supermarket*. Change the **BackgroundColor** to *White*. Click the **FontBold** check box to select it. Change the **TextAlignment** to *center* and the **FontSize** to 24.
- Delete the default **Text** for **SpacingLabel**. Change the **Height** to 10 pixels.
- Change the **Text** for **MessageLabel** to *I'm going to buy these items:.* Change the **BackgroundColor** to *White*. Click the **FontBold** check box to select it.
- Change the **Text** for **ShoppingListLabel** to *blank* (i.e. – delete the default text so this field is empty). Click the **FontBold** check box to select it.
- Change the **Hint** of **AddToListTextBox** to *Enter an item for the list*. Change the **Width** to 200 pixels.
- Change the **Text** of **AddToListButton** to *Add to List*
- Change the **Text** of **ClearOutListButton** to *Clear out the List*

## Part 2 - Program Functionality of the App

1. Change to **Blocks** view.


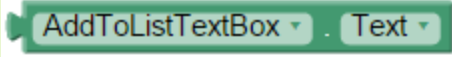
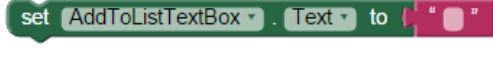
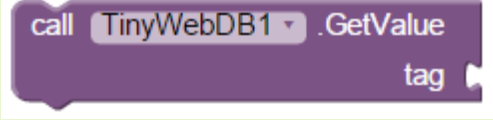
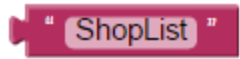

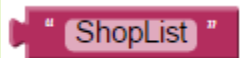

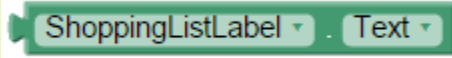
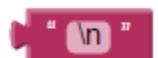
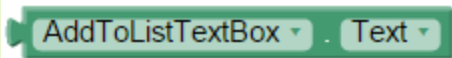
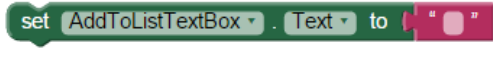
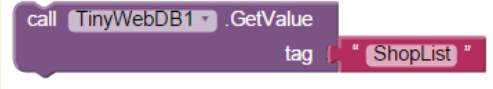
The blocks for this app are as follows:



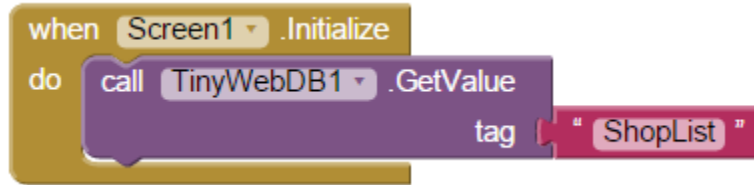
2. Now drag out the blocks (as indicated in the table below) and arrange them as shown above.

Block Type	Drawer	Purpose
	Screen1	Controls what happens when Screen1 is first initialized (the app is started)
	TinyWebDB1	Retrieves a value stored in the database that is referenced to the tag provided
	Text	Text box that holds the name of the tag associated with the data to be stored in the database.
	TinyWebDB1	Controls actions performed after a  block successfully retrieves data from the database
	ShoppingListLabel	Sets the label text to the value provided
	Obtained by pointing to the  argument in the  block	Contains the value of the data retrieved from the database
	ClearOutListButton	Controls the events after the user clicks and releases the ClearOutListButton

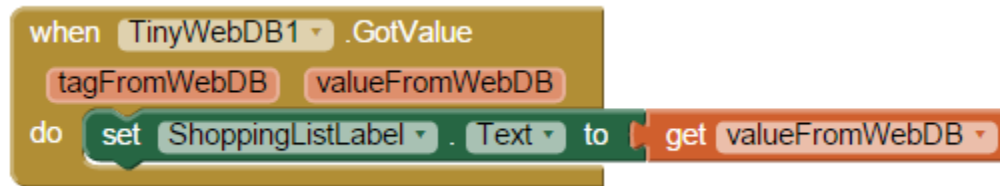
	TinyWebDB1	Asks the web service to store the given value under the given tag in the database
	Text	Text box that holds the name of the tag associated with the data to be stored in the database
	Text	Text box that holds the string (value) of the data to be stored in the database (in this case...nothing)
	TinyWebDB1	Retrieves the value stored in the database that is referenced to the given tag. In this instance, it retrieves the blank value just stored.
	Text	Name of the tag associated with the data to be retrieved from the database
	AddToListButton	Controls the events after the user clicks and releases the AddToListButton
	Control	If a value is true, do the first block of statements (then). Otherwise, do the second block (else)
	Logic	Tests if two things (values or strings) are equal
	ShoppingListLabel	Contains the value of the ShoppingListLabel
	Text	A blank (null) string
The following blocks are executed ( <i>then</i> ) if the ShoppingListLabel is currently blank (i.e. – there is nothing on the shopping list)		
	TinyWebDB1	Asks the web service to store the given value under the given tag in the database

	Text	Name of the tag associated with the data to be stored in the database
	AddToListTextBox	Contains the value of the AddToListTextBox
	AddToListTextBox, Text	Sets the value of the AddToListTextBox to "blank"
	TinyWebDB1	Retrieves the value stored in the database that is referenced to the given tag. In this instance, it retrieves the value of the current shopping list.
	Text	Name of the tag associated with the data to be retrieved from the database
The following blocks are executed ( <i>else</i> ) if the ShoppingListLabel is <i>not</i> blank (i.e. – there are items on the shopping list)		
	TinyWebDB1	Asks the web service to store the given value under the given tag in the database
	Text	Name of the tag associated with the data to be stored in the database
	Text	Appends all inputs to create a single text string (in this case, the shopping list)
	Shopping List Label	Contains the current items on the shopping list
	Text	The string <code>\n</code> indicates that a new line should be started
	AddToListTextBox	Contains the new item to be added to the current list.
	AddToListTextBox, Text	Sets the value of the AddToListTextBox to "blank"
	TinyWebDB1	Retrieves the value stored in the database referenced by the "ShopList" tag (now the current list)

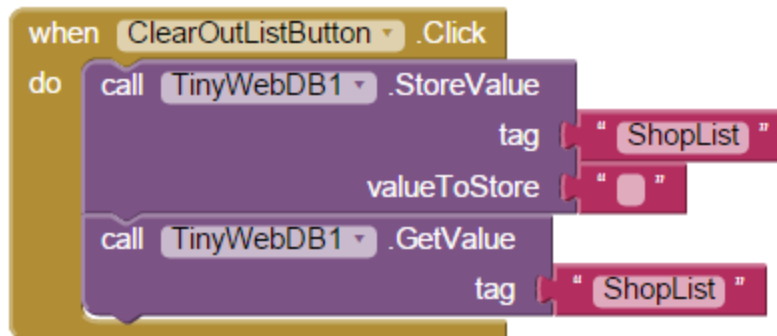
Let's look briefly at each finished block and consider what the block does.



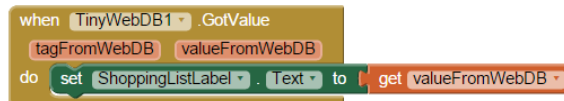
When Screen1 is initialized (the app is started), the value of the current shopping list is retrieved from storage in the database. There may be nothing on the list, or there may be items added the last time the app was run (or was run by someone else on another device).



Whenever a value is successfully retrieved from the web database, this block executes and copies the current value of the list into the ShoppingListLabel.

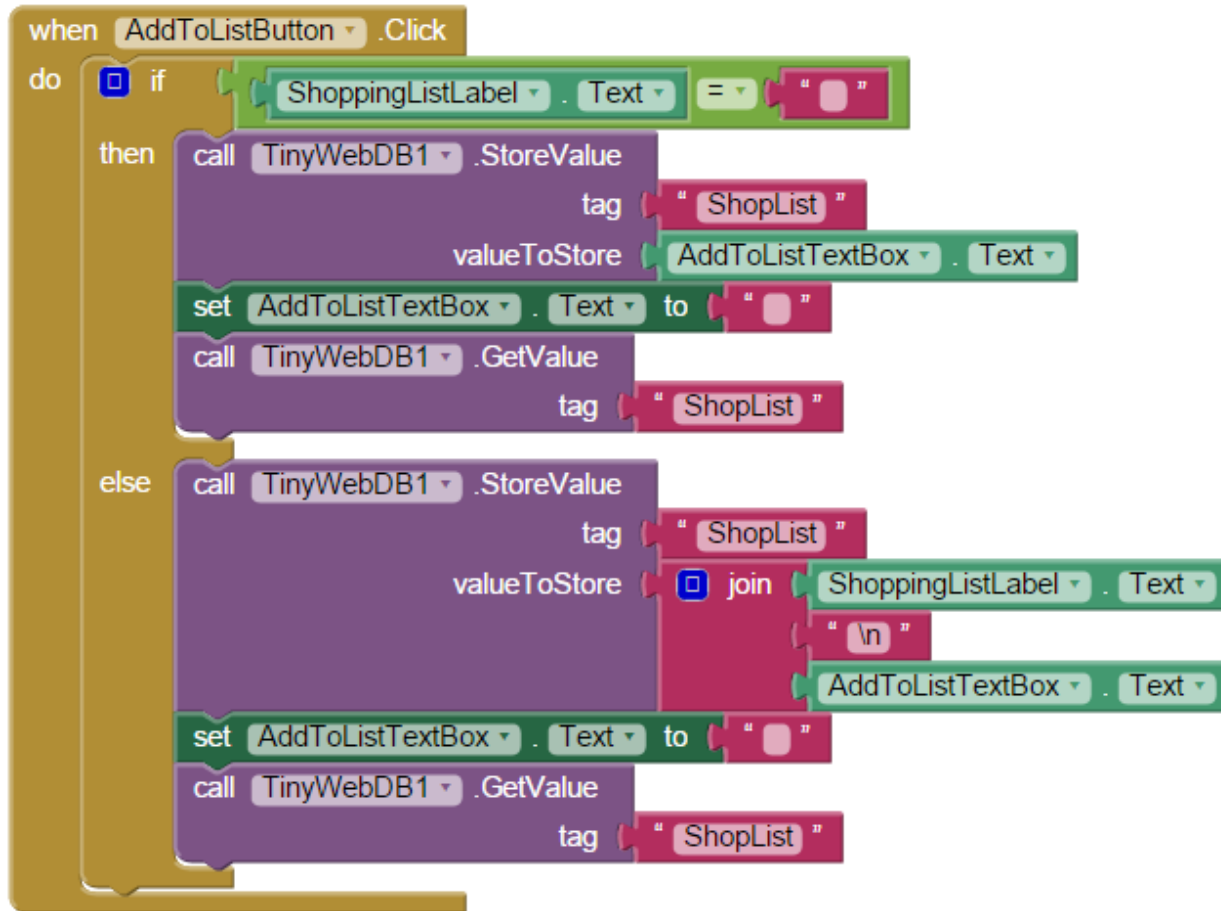


When the ClearOutListButton is clicked, the list needs to be cleared completely. A “blank” value is stored in the database for the tag “ShopList”, then this value is retrieved from the database.

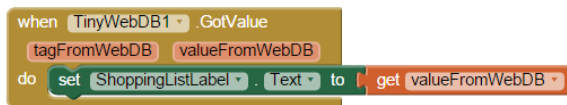


Successfully retrieving the value causes the block to execute and the value of the ShoppingListLabel becomes “blank”.



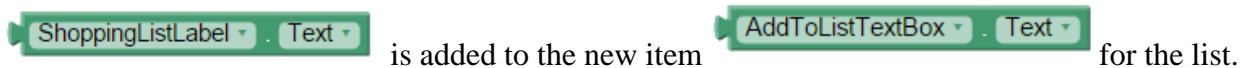


If the ShoppingListLabel is currently blank, the statements in the “then” portion of this block are executed. The value in the AddToListTextBox (the only item on the list) is stored in the database and the AddToListText box is then “blanked out” to allow for the next entry to the list. The value of the list is retrieved from the database which causes the



block to execute and fill in the ShoppingListLabel with the current value of the shopping list.

However, if the ShoppingListLabel is not blank (i.e. – there are items on the list), the statements in the “else” portion of the block are executed. The current value of the list

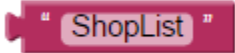


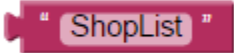
The new item is added to the bottom of the list on a new line because of the block containing the new line command (\n).



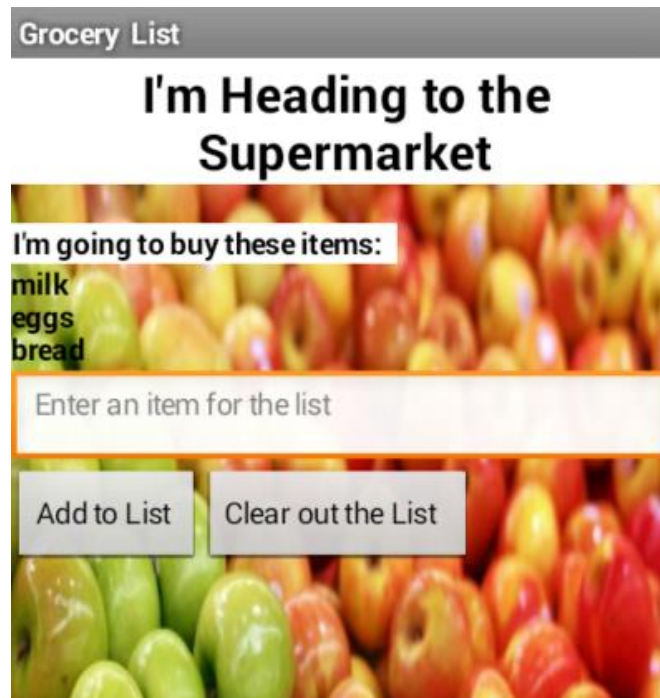
### Part 3 - Prepare the App for Testing

Before you test your app, you need to go back and make a few adjustments. The default setting for TinyWebDB is to store data on a test web service provided by App Inventor. On your App Inventor interface, if you look in the properties window (in Designer view) for TinyWebDB1, you will notice the **ServiceURL** is set to *http://appinvtinywebdb.appspot.com/*. This service is shared by all App Inventor users, including your classmates! Since you are all saving

information using the same tag  there are bound to be conflicts and overwritten information! To adequately test your app, go back into the blocks view and replace

all the  blocks with blocks that have a unique tag (perhaps your student ID number, or your first and last name, or a random string of 12 numbers). Make sure it is a unique value that your classmates wouldn't be using! Once this is done, you can test out your app!

Now test out your app by adding a few items to the list. Your screen should look like this:



Now close your app and restart it. The items you previously saved to the list should appear when the app initializes. Now clear the list and add a few new items.

## Extensions to This Project

---

1. Perhaps you want to see if your app works on one of your classmates' devices– so your classmate can add items to your shopping list. Your classmate will first need to install your app. First, you need to “package” it up as an .apk file so they can install it on their Android device. The instructions for packaging apps is found [here](#).
2. If you are creating an app for distribution, you don't want it to reference the default web service that TinyWebDB uses for testing. You need to create a custom TinyWebDB service for your own use. Review the documentation for doing this [here](#). Follow the instructions and set up you own custom web service. Then modify your app to use that service.

## Resources

---

- [AI2 Storage Components: TinyWebDB](#)
- [Creating a Custom TinyWebDB Service](#)
- [User Guide for App Inventor 2](#)
- [Guide to Understanding Blocks](#)

MIT App Inventor is a blocks-based programming tool that allows everyone, even novices, to start programming and build fully functional apps for Android devices. Google's Mark Friedman and MIT Professor Hal Abelson co-led the development of App Inventor while Hal was on sabbatical at Google. App Inventor runs as a Web service administered by staff at MIT's Center for Mobile Learning - a collaboration of MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) and the MIT Media Lab. MIT App Inventor supports a worldwide community of nearly 3 million users representing 195 countries worldwide. App Inventor is an open-source tool that seeks to make both programming and app creation accessible to a wide range of audiences. App Inventor is the property of the Massachusetts Institute of Technology (MIT) and the work licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). For more information on App Inventor, go to [MIT App Inventor About Us page](#).