User Interface Design and Application

AngularJS UI Grid

Final Project

Alexander R Ramsey

University of Maryland University College

Abstract

Web Application User Interfaces have evolved in the last decade to allow for asynchronous communication with web services. The challenge of developing interactive user interfaces, while integrating with legacy systems remains a challenge in the enterprise application development community. PHP, a server side scripting language used in web development has now evolved over two decades to become a major partner of the MySQL open source database technology. In this document, a simple web application user interface is proposed as an accessor to a legacy PHP and MySQL server side. AngularJS, a modern front end framework built on the JavaScript language is used to communicate with PHP through a Representational State Transfer (REST) Application Platform Interface (API). Code examples are pulled from the web for the server side, so that development efforts can be focused in on the UI development.

*Keywords:* AngularJS, PHP, MySQL, AngularJS UI-Grid, REST API

Introduction

This document outlines the process of user interface development, implementation of a server side scripting language, and a Create, Read, Update, Delete (CRUD) driven MySQL database. An application for a local business will be implemented to manage customer data through a simple user interface. The implementation of the user interface design will be attempted through a code exploration.

Tech Stack

PHP, MySQL, Bootstrap, AngularJS Single Page Application (SPA), AngularJS UI-Grid

Data Model

The Data Model used for this application will be represented as a Customer entry with the following data fields: Customer Name, Email, Address, City, State, Postal Code, and Country.

User Interface Development

*A personal experience with Wireframe.cc can be viewed here:* [*https://youtu.be/3jcGe9HAuvI*](https://youtu.be/3jcGe9HAuvI)

The concept behind this application is to provide a simple interface for an office employee to interact with on a call to call basis. While interacting with new clients and existing clients, the employee should be able to easily review and enter information with minimal effort or previous experience to the application.

To accomplish this, a user interface that allows ease of movement between operations will be implemented. This will include access to a grid that displays pertinent information. For the scope of the project, a table will be implemented focusing solely on the customer data.

The proposed User Interface will be encompassed by a Single Page Application (SPA). This will allow the user to interact with a single page for which they can perform CRUD operations within a User Interface Grid / Table View Layout.

The table view layout will include form interactivity so that a user can rapidly update customer information from within the data grid. Using a Wireframe browser application (http://wireframe.cc), one can quickly mockup some ideas like below:

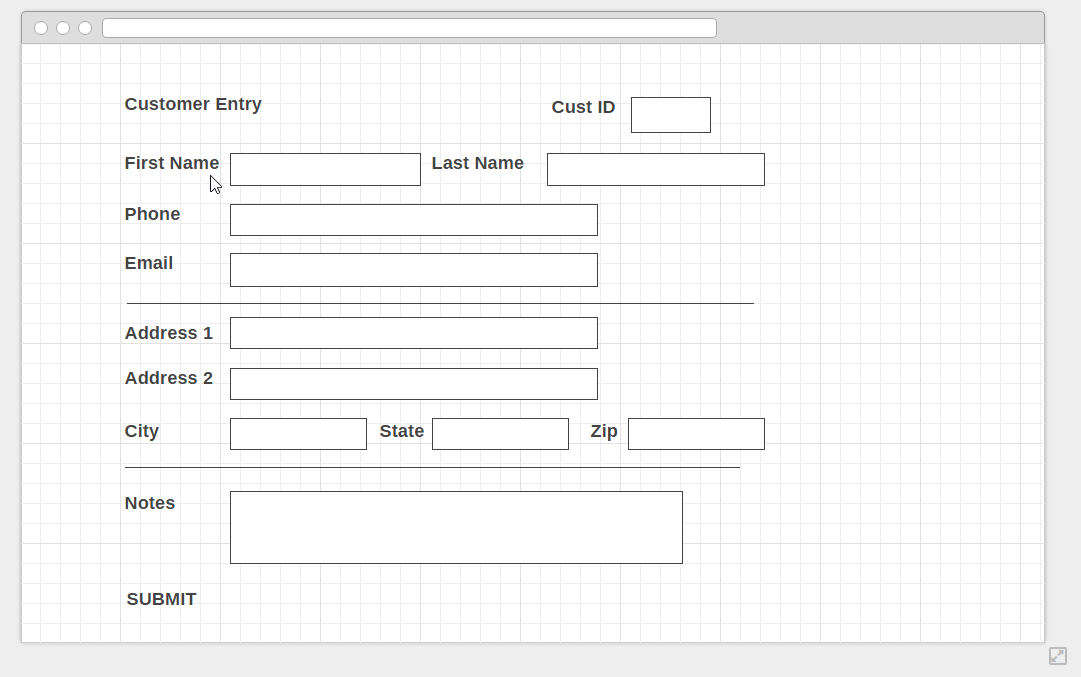


Figure 1 An edit view of a Customer entry record.

This wireframe concept is an example of an Edit View which can be used to update existing records stored within the database. A First Name, Last Name, Email, Address, City, State, Postal Code, and Country will be the primary focus of the application data.

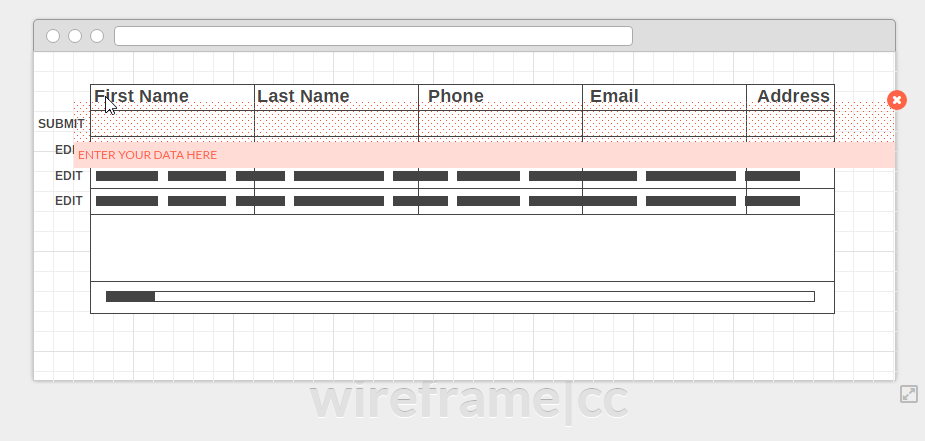


Figure 2 A data grid view of existing customer entries. Annotation reads, “Enter your data here” overtop of blank data grid row form fields which allow a user to input data.

In the table grid view, a user would be able to view table data. In addition, a user would be able to insert new table data into data fields nested at the top of the table. A submit button would allow the user to send the data to the database. An edit button would allow the user to edit an existing record with the opening of a new view like the edit view seen in the first wireframe mockup figure. This view could be easily made into a JavaScript modal which then calls a database update function upon submission.

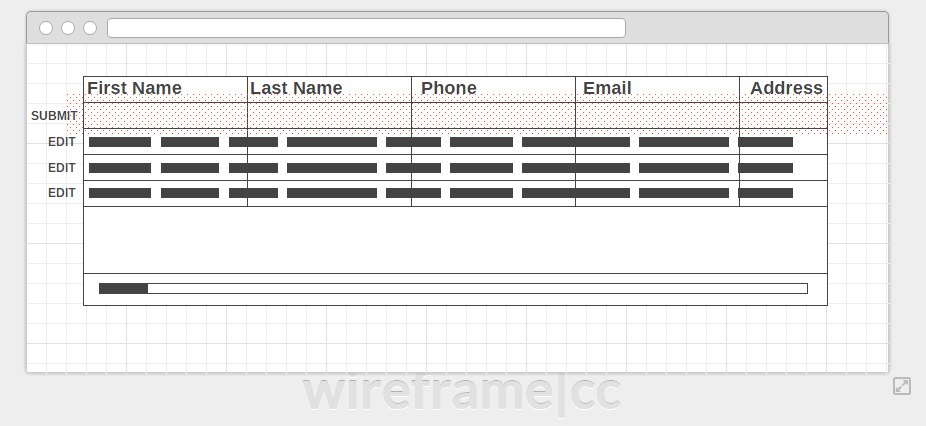


Figure 3 Existing data is displayed beneath the submit data form field row of the table.

Tech Stack Initialization

Server Side Technologies

A handy [tutorial from AngularCode.com](http://angularcode.com/demo-of-a-simple-crud-restful-php-service-used-with-angularjs-and-mysql/) describes using PHP as a RESTful API with a modern AngularJS front end. A live demo of the application tutorial can be seen [here](http://demos.angularcode.com/AngularCodeCustomerManagerApp/#/edit-customer/20532).

Client Side Technologies

UI Grid is a platform in AngularJS that supports the extensibility of data manipulation like sorting, filtering, expansion of rows, and nearly any functionality surrounding a user manipulating a table of data. We’ll deploy this new technology into the existing provided PHP framework. While doing so, we’ll look at Design Patterns and Design Anti Patterns to make code more efficient.

Development Environment Setup Steps

1. WAMP Stack is used locally to test before deployment to a server for testing.
2. REWRITE\_MOD is enabled in Apache Server to support URL routing.
3. Then extraction of the example application starter files to the local server working directory.
4. Open the angularcode\_customer.sql file
5. Execute the contents into a MySQL console in PHPMyAdmin. This generates the appropriate database.
6. Modify the connection string in the /Services/Api.php file to connect to your local database. Defaults:

const DB\_SERVER = "localhost";

const DB\_USER = "root";

const DB\_PASSWORD = "";

const DB = "angularcode\_customer";

Note: You may need to append a port number to the end of localhost for database connection to be successful. Example: localhost:3306

1. Navigate browser to localhost/AngularCodeCustomerManagerApp/ and voila:

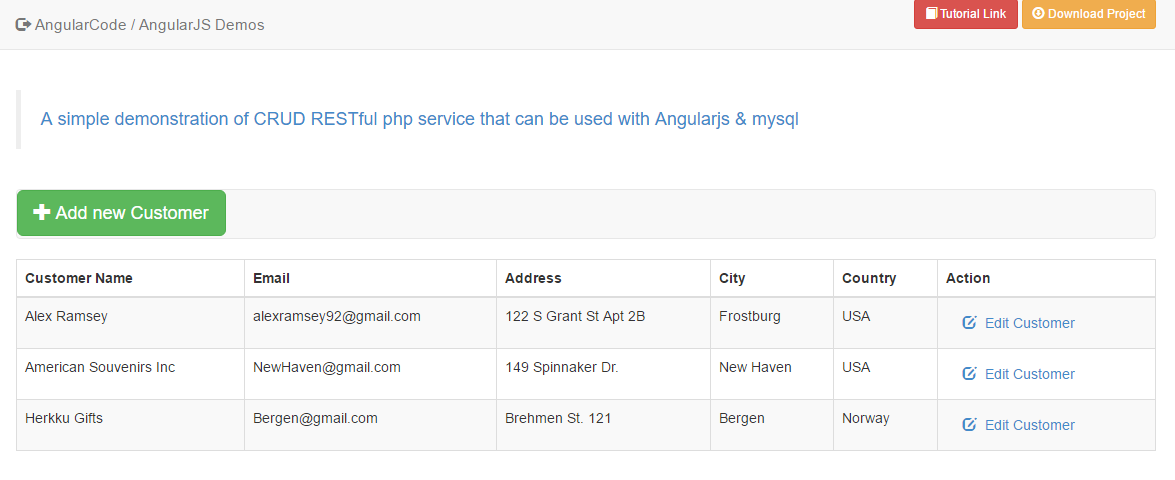


Figure 4 The homepage of the locally hosted example code from AngularCode.com with original header removed.

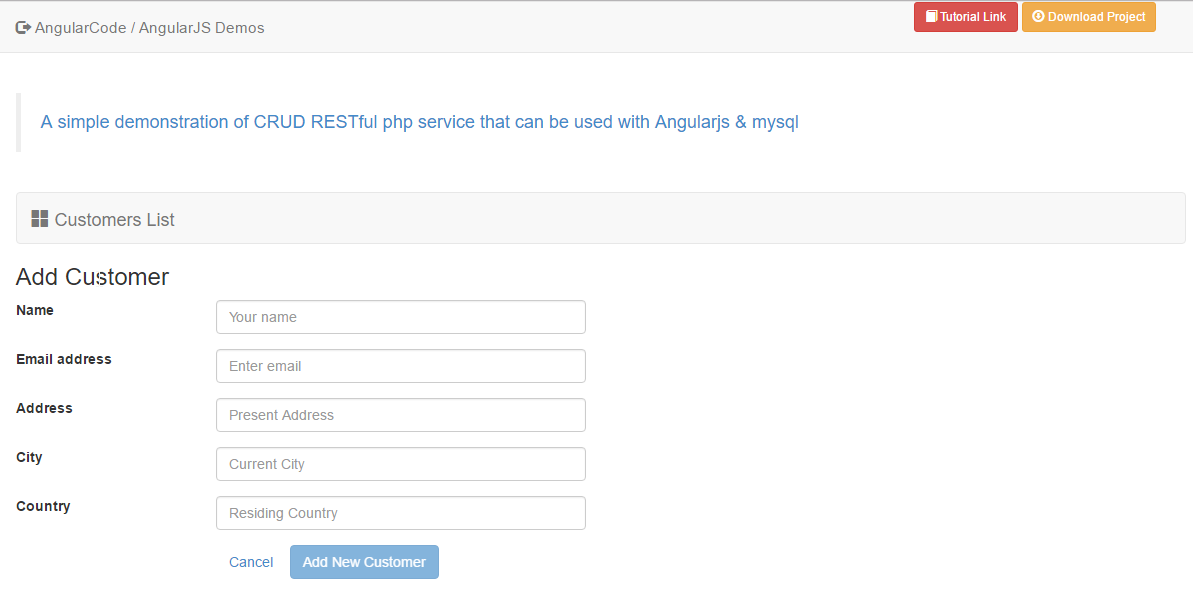


Figure 5 The edit customer view of the locally hosted example code from AngularCode.com with original header removed.

Displayed above is a REST-API web application connected to a MySQL database with AngularJS communicating through a REST API powered by PHP. All on a local WAMP stack. An interesting blend of older MYSQLI database connectivity syntax in PHP and a new JavaScript framework called AngularJS.

Now focus can be shifted onto the UI-Grid package of AngularJS. To get started with the integration of the server side scripting and the Angular UI-Grid, a new Template view and /uigrid URL rule for Angular are created. The route provider in AngularJS acts as a convenient way to specify template views, abstracting the individual user interface functions away from the main webpage styles, header, footer, and dependency calls to framework files.

app**.**config**([**'$routeProvider'**,**

***function*(**$routeProvider**)** **{**

$routeProvider**.**

when**(**'/'**,** **{**

title**:** 'Customers'**,**

templateUrl**:** 'partials/customers.html'**,**

controller**:** 'listCtrl'

**})**

**.**when**(**'/edit-customer/:customerID'**,** **{**

title**:** 'Edit Customers'**,**

templateUrl**:** 'partials/edit-customer.html'**,**

controller**:** 'editCtrl'**,**

resolve**:** **{**

customer**:** ***function*(**services**,** $route**){**

***var*** customerID **=** $route**.**current**.**params**.**customerID**;**

***return*** services**.**getCustomer**(**customerID**);**

**}**

**}**

**})**

**.**when**(**'/uigrid'**,** **{**

templateUrl **:** 'partials/uigrid.html'**,**

controller **:** 'uiGridCtrl'

**})**

**.**otherwise**({**

redirectTo**:** '/'

**});**

**}]);**

Figure 6 Code snippet: The AngularJS route provider allows for quick and easy URL structure and view templating within the project.

The route provider shown above calls a template view in the /partials folder called uigrid.html and executes instructions in the controller, uiGridCtrl.

Controllers to Perform CRUD Operations

The uiGridCtrl acts as a starting point for the development of the user interface CRUD functions. For each operation, we will explore the evolution of the controller and other code elements required to initiate functionality.

Read operation

app**.**controller**(**'uiGridCtrl'**,** ***function*** **(**$scope**,** services**)** **{**

services**.**getCustomers**().**then**(*function*(**data**){**

$scope**.**customers **=** data**.**data**;**

**});**

**});**

With a working route provider and Read operation within the controller, we can navigate our browser to localhost/AngularCodeCustomerManagerApp/#/uigrid   
We get a working read only implementation of the Angular JS UI-Grid.

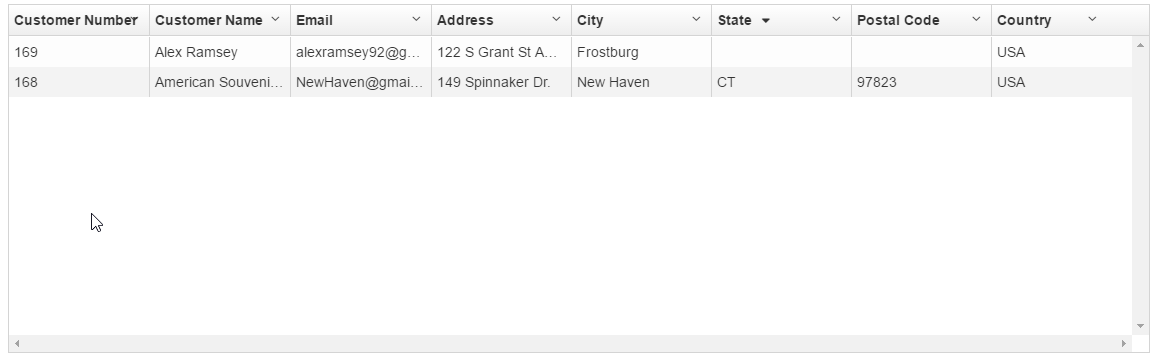


Figure 7 Angular UI-Grid Read only table after integration with PHP REST API.

Initial setup of AngularJS UI-Grid is successful, providing read only access to the table. However, some problems appear during implementation. The client requests that we must have a scalable table that will display well on any device. Unfortunately, after researching the AngularJS UI-Grid capabilities, it is found that the UI-Grid library does not easily support a fully responsive and browser width friendly table, even for bigger displays. After many attempts to set the width of the table and to adjust the sizing of elements, focus is redirected back to the original table in the example tutorial. The question arises, why not adapt it to suit our interface needs?

Adaptation of Original Table View

Being that CRUD operation functionality already exists for this example, let’s adapt the HTML and JS Controller to extend the list view to support the creation of new records, Updating of Existing Records, and Deletion of Records. These operations will be in line to the page and will not require that the user divert their attention from the existing user interface layout.

We can adapt the existing listCtrl in the Angular Route Provider to accommodate for newly added functionality by using the following code:

app**.**controller**(**'listCtrl'**,** ***function*** **(**$scope**,** $rootScope**,** $location**,** $routeParams**,** services**)** **{**

$scope**.**newcustomer **=** **{};**

services**.**getCustomers**().**then**(*function*(**data**){**

$scope**.**customers **=** data**.**data**;**

**});**

$scope**.**refresh **=** ***function*()** **{**

services**.**getCustomers**().**then**(*function*(**data**){**

$scope**.**customers **=** data**.**data**;**

**});**

**}**

$scope**.**isClean **=** ***function*()** **{**

***return*** angular**.**equals**(**original**,** $scope**.**customer**);**

**}**

$scope**.**deleteCustomer **=** ***function*(**customer**,** $index**)** **{**

***if*(**confirm**(**"Are you sure to delete customer number: "**+**customer**.**customerNumber**)==*true*)**

services**.**deleteCustomer**(**customer**.**customerNumber**).**then**(*function*(**$data**){**

$scope**.**customers**.**splice**(**$index**,** 1**);**

**},*function*(**$err**){**

**})**

**};**

$scope**.**insertCustomer **=** services**.**insertCustomer**;**

$scope**.**updateCustomer **=** services**.**updateCustomer**;**

**});**

And, we can adapt the Partials/Customer.html code to implement the CRUD functions:

**<!--** Adapted by Al Ramsey ***for*** SWEN 651 **-->**

**<**table ***class*=**"table table-striped table-bordered responsive"**>**

**<**thead**>**

**<**th**>**Customer Name**&**nbsp**;</**th**>**

**<**th**>**Email**&**nbsp**;</**th**>**

**<**th**>**Address**&**nbsp**;</**th**>**

**<**th**>**City**&**nbsp**;</**th**>**

**<**th**>**Country**&**nbsp**;</**th**>**

**<**th**>**Action**&**nbsp**;</**th**>**

**</**thead**>**

**<**tbody**>**

**<**tr ng**-**repeat**=**"customer in customers" ng**-**show**=**"customers.length > 0"**>**

**<!--** replace ***with*** a directive at some point **-->**

**<**td ng**-**hide**=**"customer.isEditing"**>{{**customer**.**customerName**}}</**td**>**

**<**td ng**-**hide**=**"customer.isEditing"**>{{**customer**.**email**}}</**td**>**

**<**td ng**-**hide**=**"customer.isEditing"**>{{**customer**.**address**}}</**td**>**

**<**td ng**-**hide**=**"customer.isEditing"**>{{**customer**.**city**}}</**td**>**

**<**td ng**-**hide**=**"customer.isEditing"**>{{**customer**.**country**}}</**td**>**

**<**td ng**-**hide**=**"customer.isEditing"**>**

**<**button ng**-**click**=**"customer.isEditing=true" ***class*=**"btn btn-link"**>**Edit**</**button**>**

**<**button ng**-**click**=**"deleteCustomer(customer, $index)" ***class*=**"btn btn-link"**>**Delete**</**button**>**

**</**td**>**

**<**td ng**-**show**=**"customer.isEditing"**><**input ng**-**model**=**"customer.customerName"**></**td**>**

**<**td ng**-**show**=**"customer.isEditing"**><**input ng**-**model**=**"customer.email"**></**td**>**

**<**td ng**-**show**=**"customer.isEditing"**><**input ng**-**model**=**"customer.address"**></**td**>**

**<**td ng**-**show**=**"customer.isEditing"**><**input ng**-**model**=**"customer.city"**></**td**>**

**<**td ng**-**show**=**"customer.isEditing"**><**input ng**-**model**=**"customer.country"**></**td**>**

**<**td ng**-**show**=**"customer.isEditing"**>**

**<**button ng**-**click**=**"updateCustomer(customer, $index); customer.isEditing=false;" ***class*=**"btn btn-link"**>**Save**</**button**>**

**<**button ng**-**click**=**"refresh();" ***class*=**"btn btn-link"**>**Cancel**</**button**>**

**<**button ng**-**click**=**"deleteCustomer(customer, $index)" ***class*=**"btn btn-link"**>**Delete**</**button**>**

**</**td**>**

**</**tr**>**

**<**tr**>**

**<**td**><**input ng**-**model**=**"newcustomer.customerName"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.email"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.address"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.city"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.country"**></**td**>**

**<**td**>**

**<**button ng**-**click**=**"insertCustomer(newcustomer); customers.push(newcustomer); newcustomer={}; refresh();" ***class*=**"btn btn-link"**>**Add**</**button**>**

**<**button ng**-**click**=**"refresh();" ***class*=**"btn btn-link"**>**Cancel**</**button**>**

**</**td**>**

**</**tr**>**

**</**tbody**>**

**</**table**>**

**<**div ***class*=**"col-md-12" ng**-**show**=**"customers.length == 0"**>**

**<**div ***class*=**"col-md-12"**>**

**<**h4**>**No customer records found**.** Enter record details and Click Add**.</**h4**>**

**</**div**>**

**</**div**>**

Although not the most scalable implementation, we can see that the simple ng-model, ng-show, ng-hide, and ng-click AngularJS directives can be used in this scenario to quickly implement show/hide, model data, and controller action functions to implement the desired CRUD functionality. What we finish with is an elegant and interactive single view table with customer data, which can be manipulated without having to move away from the original user interface.

Figure 8 After modification of the existing table view, with the implementation of Angular directives, the user can perform all CRUD operations in a single view.

Almost done, now some simple prompts and data validation can be added to the input fields in the table to ensure the user provides the correct data. This is accomplished by using AngularJS directives like ng-pattern and HTML5 input, placeholder and, type tags.

**<**tr**>**

**<**td**><**input ng**-**model**=**"newcustomer.customerName" ***class*=** "form-control" placeholder**=**"Enter Name"**></**td**>**

**<**td**>**

**<**input type**=**"email" name**=**"email" ng**-**model**=**"newcustomer.email" ***class*=**"form-control" placeholder**=**"Enter Email" ng**-**pattern**=**'/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/' required**/>**

**<**div role**=**"alert"**>**

**<**span ***class*=**"error" ng**-*if*=**"form.email.$error.required"**>**

Required**!</**span**>**

**</**div**>**

**</**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.address" ***class*=** "form-control" placeholder**=**"Enter Address"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.city" ***class*=** "form-control" placeholder**=**"Enter City"**></**td**>**

**<**td**><**input ng**-**model**=**"newcustomer.country" ***class*=** "form-control" placeholder**=**"Enter Country"**></**td**>**

**<**td**><**div ***class*=**"btn-group"**>**

**<**button ng**-**click**=**"insertCustomer(newcustomer); customers.push(newcustomer); newcustomer={}; refresh();" ***class*=**"btn btn-success" ng**-**disabled**=**"form.$invalid"**>**Add**</**button**>**

**<**button ng**-**click**=**"refresh();" ***class*=**"btn btn-normal"**>**Cancel**</**button**>**

**</**div**>**

**</**td**>**

**</**tr**>**

Concluding Remarks

The original application aim was to consolidate CRUD operations into a single user interface, while using a modern responsive JavaScript Framework called AngularJS. Overall, the project is successful. Some bugs remain involving form validation and the transfer of front end day to the AngularJS array. In the end, a lot of ground was covered in the exploration of developing a user interface from concept to reality.

References

Anonymous Contributor. AngularCode.com. Demo of a simple CRUD RESTful php service used with AngularJS and MySQL. (n.d.) Retrieved from <http://angularcode.com/demo-of-a-simple-crud-restful-php-service-used-with-angularjs-and-mysql/>