



# DrupalGap

A Mobile Application Development Kit for Drupal

## Presentation for DrupalCamp Michigan 2013 in Ann Arbor

- What is DrupalGap?
- How does it Work?
- How to Build a Mobile App for a Drupal Website?

*by:* Tyler Frankenstein

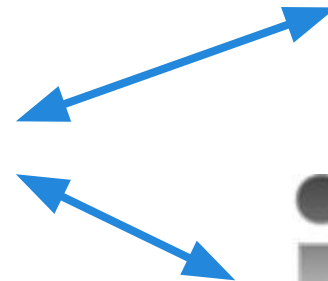
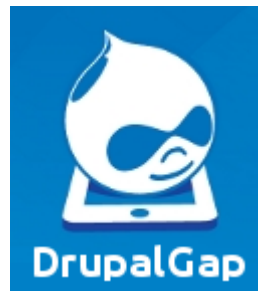


**EasyStreet3**  
MOBILE APPS & WEBSITES



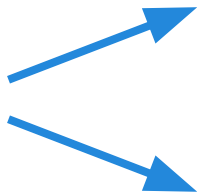
# What is DrupalGap?

An Open Source Mobile Application Development Kit  
for Drupal Websites



Build Custom Mobile Applications for Android and iOS  
Devices to Communicate with Drupal Websites

# DrupalGap has two parts...



Drupal Module

Mobile Application Development Kit

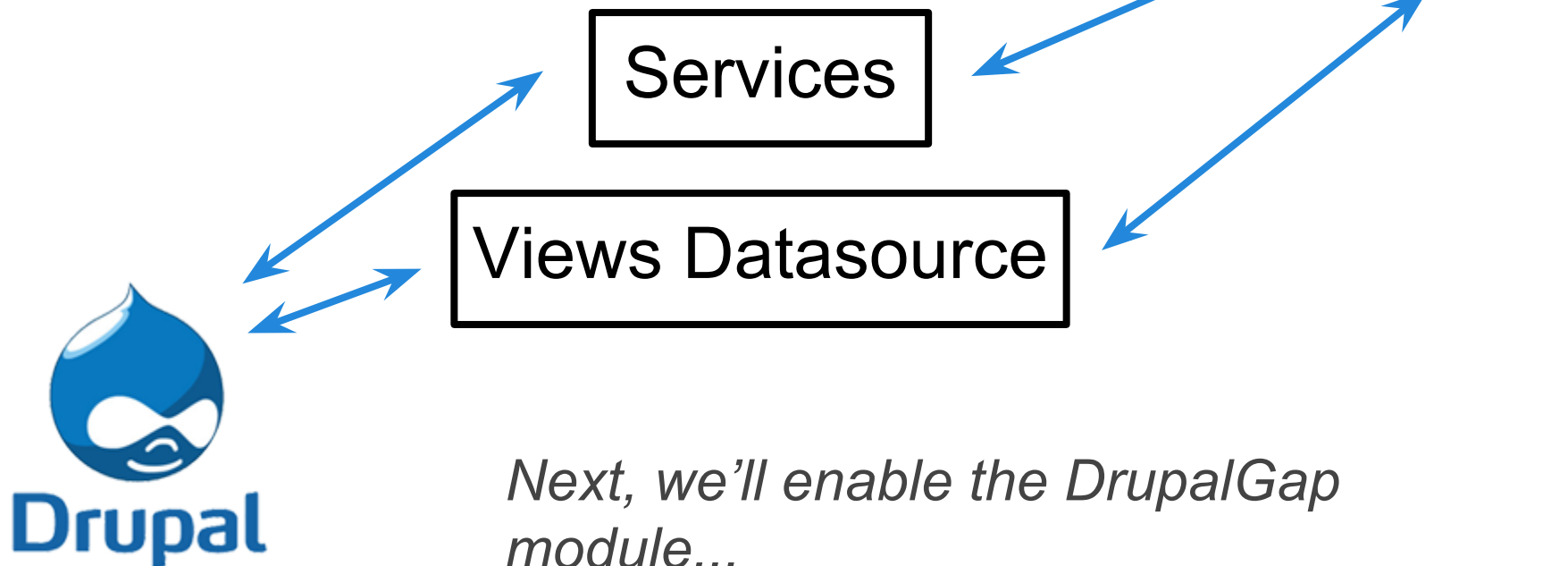
- <http://www.drupal.org/project/drupalgap>
- <https://www.github.com/signalpoint/DrupalGap>

The **Drupal Module** opens up communication possibilities between a mobile app and the website. The **Mobile Application Development Kit** is used by developers to create custom mobile applications.

*Let's first look at the DrupalGap module...*

# How does the DrupalGap module work?

The DrupalGap module uses the **Services** and **Views Datasource** modules to help mobile apps communicate with a Drupal website using JSON.



# To Get Started, Enable the DrupalGap Module on a Drupal Website

- **DrupalGap** module
  - <http://www.drupal.org/project/drupalgap>

## *Dependencies:*

- **REST Server** module
  - (available in the **Services module**)
  - <http://www.drupal.org/project/services>
- **Views JSON** module
  - (available in the **Views Datasource module**)
  - [http://www.drupal.org/project/views\\_datasource](http://www.drupal.org/project/views_datasource)

*For starters, we just need to enable the DrupalGap module.  
Then let's jump right into mobile application development...*

# What tools does the DrupalGap mobile application development kit use to build apps?



DrupalGap uses jQueryMobile to build the user interface and PhoneGap to compile the App for installation on Android and iOS mobile devices.



*Let's first look at how PhoneGap works...*

# How does PhoneGap work?

PhoneGap takes HTML, CSS and JavaScript and compiles it into a Mobile Application for iOS and Android mobile devices.

PhoneGap provides access via JavaScript to device features such as the Camera, GPS, File System, Contacts, Compass, etc.

If we can build a website, then we can build a mobile app for multiple platforms, with one set of code...

HTML + CSS + JavaScript



PhoneGap

ios



# A Simple Mobile Application Page in PhoneGap

e.g. `~/phonegap/www/index.html`

```
<html>
<body>
  <h1>My Custom App</h1>
  <ul>
    <li><a href="#">Button #1</a></li>
    <li><a href="#">Button #2</a></li>
  </ul>
  <p>Hello World</p>
  <h2>www.example.com</h2>
</body>
</html>
```

*HTML Code has been trimmed for simplicity*



## My Custom App

- [Button #1](#)
- [Button #2](#)

Hello World

**www.example.com**

Not very impressive. Let's see how jQueryMobile can help us.



# Applying jQueryMobile to the same Mobile Application Page

By using **jQueryMobile**, a few **div containers**, and the **'data-role'** attribute, our simple page looks much better.

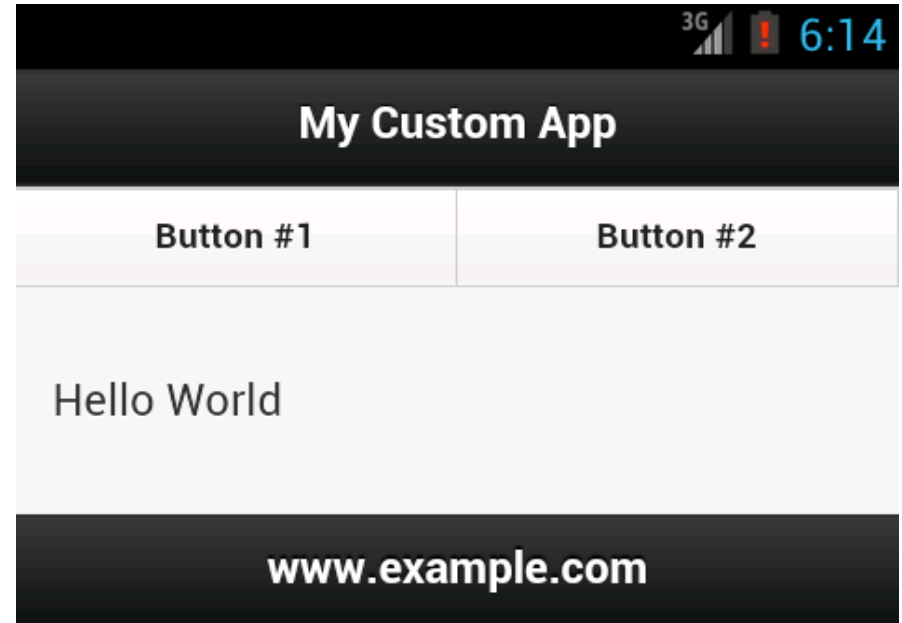


## My Custom App

- [Button #1](#)
- [Button #2](#)

Hello World

www.example.com



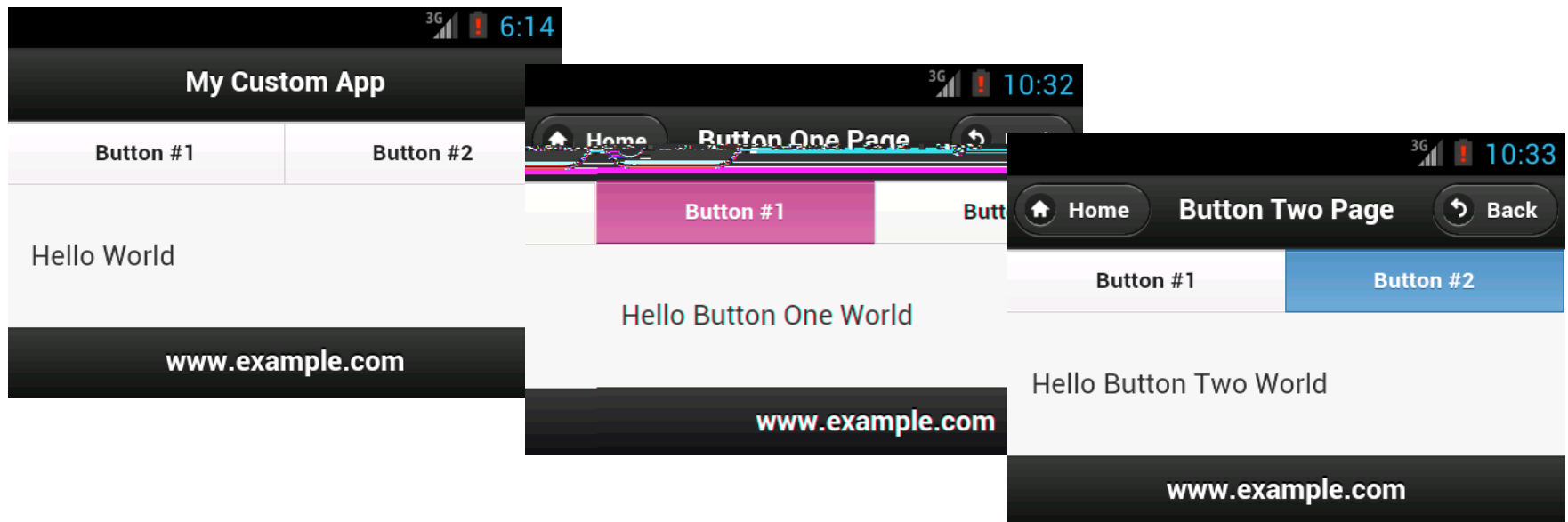
# The HTML used by jQueryMobile to Display the Mobile Application Page

```
<html>
  <head><!-- jQueryMobile includes go here --></head>
  <body>
    <div data-role="page">
      <div data-role="header"><h1>My Custom App</h1></div>
      <div data-role="navbar"><ul>
        <li><a href="#">Button #1</a></li>
        <li><a href="#">Button #2</a></li>
      </ul></div>
      <div data-role="content"><p>Hello World</p></div>
      <div data-role="footer"><h2><a href="http://www.example.com">www.example.com</a></h2></div>
    </div><!-- page -->
  </body>
</html>
```

*e.g. ~/phonegap/www/index.html*

# That's great, but...

What if we wanted to create another page in our app with the same header, navigation bar and footer, but with a different content area?



Would we copy and paste the page HTML, then modify only small chunks of HTML on the new page? We could do that, but if our app has many pages, this will become a nightmare to maintain.

*Introducing the DrupalGap Mobile Application Development Kit...*

# Let's Add DrupalGap to the Mobile App

Place the DrupalGap Mobile Application Development Kit files into **PhoneGap's www directory** to utilize its features:

- ~/phonegap/www/**index.html**
- ~/phonegap/www/**drupalgap.js**
- ~/phonegap/www/app/default.settings.js
- ~/phonegap/www/modules/\*
- ~/phonegap/www/themes/\*
- ... etc ...

*DrupalGap comes with core modules, a theme, default settings, and a place to house all custom code. The 'app' directory is similar to the 'sites/all' directory in Drupal.*

# Save a copy of `default.settings.js` as `settings.js`, then set the `site_path`

`~/phonegap/www/app/default.settings.js`



`~/phonegap/www/app/settings.js`

Then open **settings.js** and set the **site\_path** variable to your Drupal site:

```
// Site Path
```

```
drupalgap.settings.site_path = 'http://www.example.com';
```

# The DrupalGap mobile application development kit is installed, now what?



DrupalGap utilizes familiar Drupal concepts like...

- Themes
- Regions
- Blocks
- Menus
- Pages

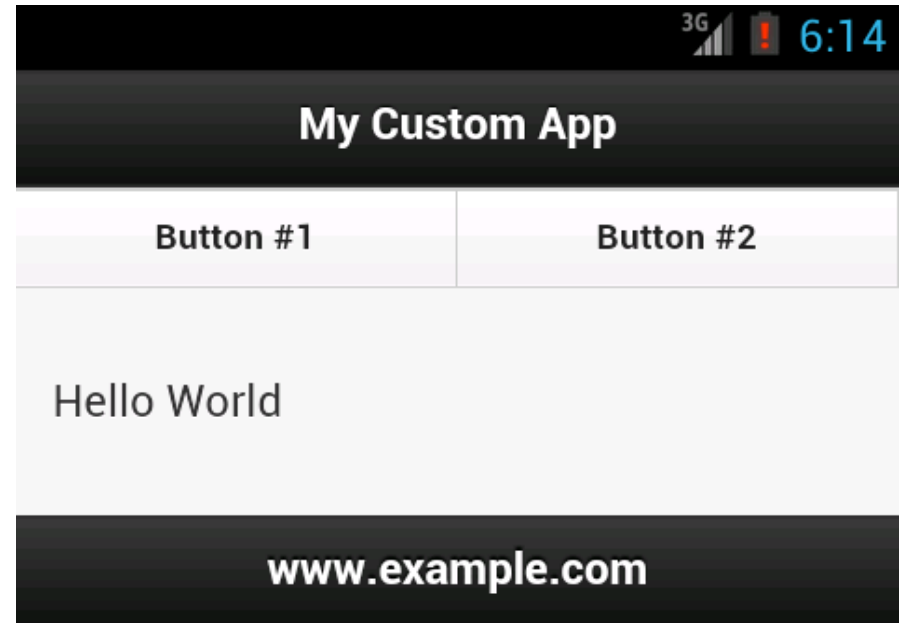
... and techniques for customization such as...

- Modules
- Hooks
- Themes
- Templates

*Next, let's examine how a page is built in DrupalGap...*

# Let's Rebuild our Example Page using DrupalGap

To create the content of this app's page, we will take an approach very similar to creating a page in **Drupal**, by using **hook\_menu()**.



But instead of using **PHP in Drupal**, we'll be using **JavaScript in PhoneGap**. Let's start by creating a custom module in DrupalGap...

# Build a custom module in DrupalGap

Add a directory and JavaScript file for the module in the DrupalGap **'app'** folder:

```
www/app/modules/custom/my_module/my_module.js
```

Now we tell DrupalGap about our custom module in the **settings.js** file:

```
// Custom Modules
```

```
drupalgap.modules.custom = [  
  {name:'my_module'},  
];
```

*For the next step, we'll implement **hook\_menu()** and add a **callback** function to handle the page's content display using JavaScript...*



```
/**  
 * Implements hook_menu().  
 */  
function my_module_menu() {  
  var items = {  
    my_custom_page:{  
      title:"My Custom App",  
      page_callback:"my_module_custom_page"  
    }  
  };  
  return items;  
}
```

```
function my_module_custom_page() {  
  return {  
    my_welcome_text:{  
      markup:"<p>Hello World</p>"  
    }  
  };  
}
```

# Set the App's front page in the settings.js file

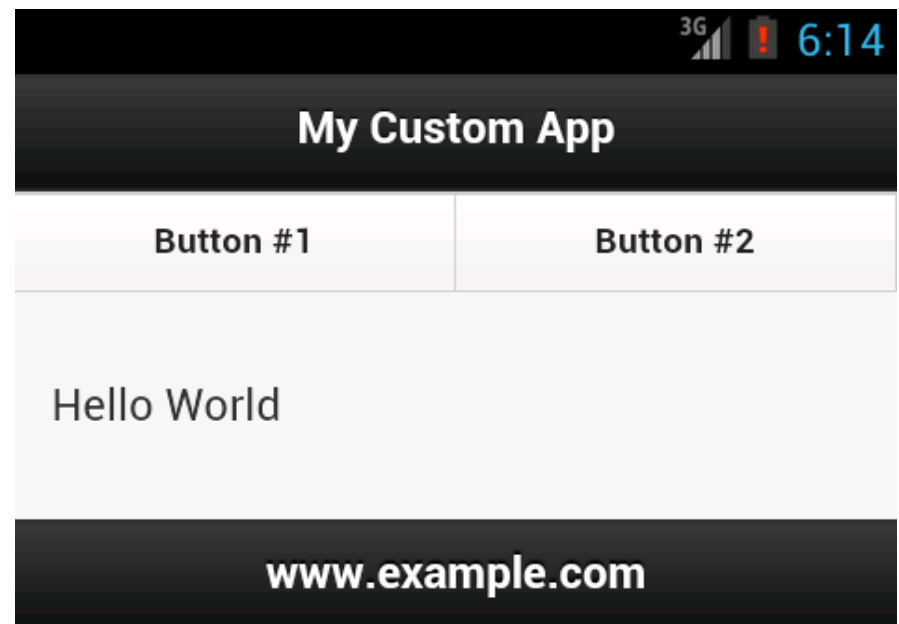
Open **settings.js** and set the **front** variable equal to the path of the newly created DrupalGap page:

```
// App Front Page
```

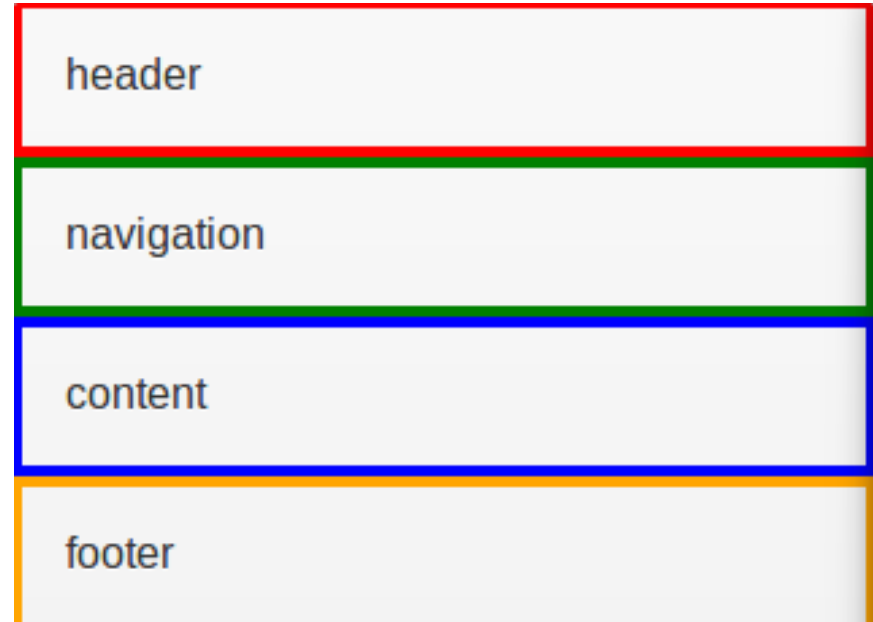
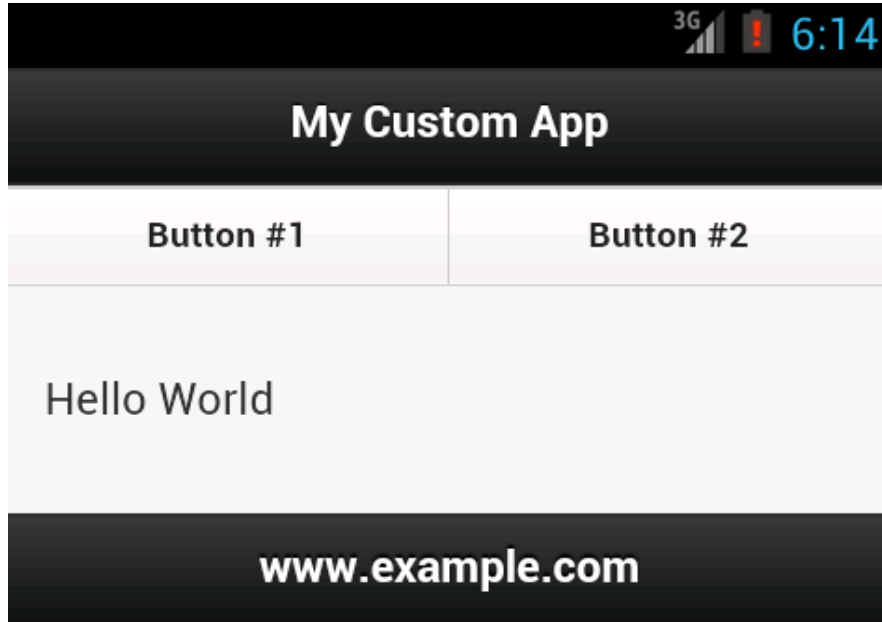
```
drupalgap.settings.front = 'my_custom_page';
```

Now if we run this DrupalGap mobile app, we'll have a page very similar to this. The page has the **title** and **body** content we specified in our custom module.

*What about the **menu buttons** and the **footer text**, where did they come from?*



# DrupalGap uses Regions in a Theme to construct a Page Template, just like Drupal

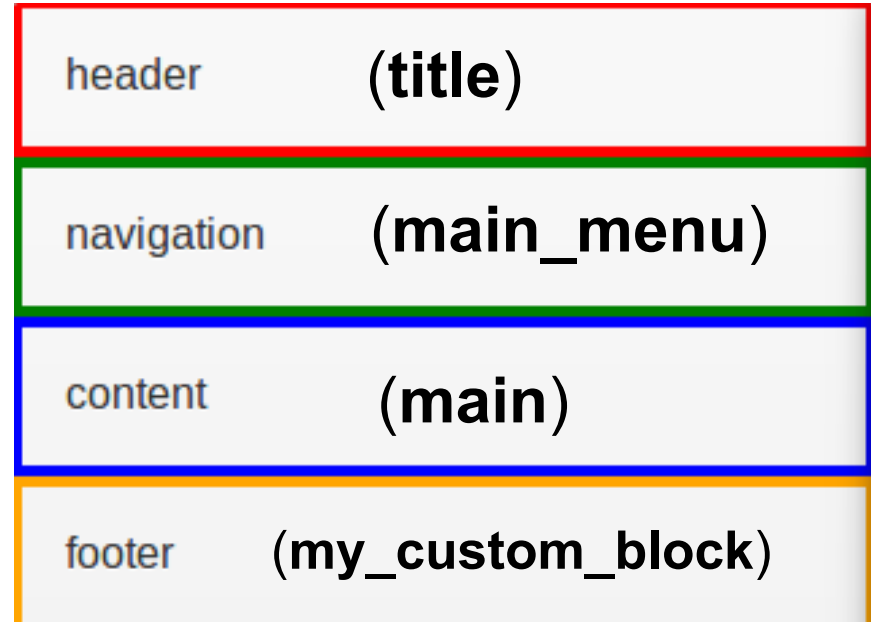
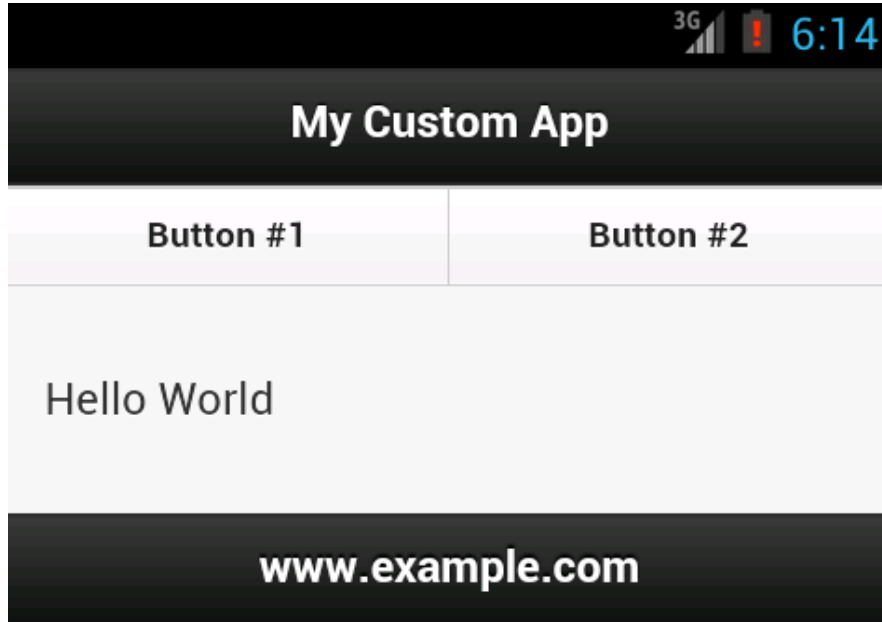


Our example page can be broken apart into 4 **Regions** to build a **Page Template** for the **Theme**.

For simplicity, we'll just use the **theme** (*easystreet3*) provided with DrupalGap core. It has a **page template** and **regions** we can use.

*More info on custom themes: <http://www.drupalgap.org/node/84>*

# DrupalGap also uses Drupal's concept of Blocks to build the HTML output for each Region

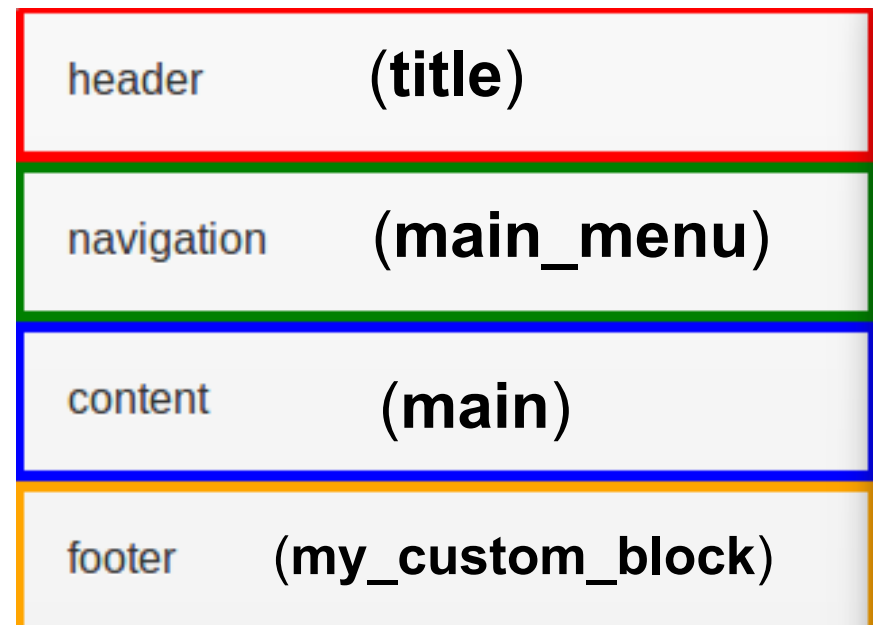
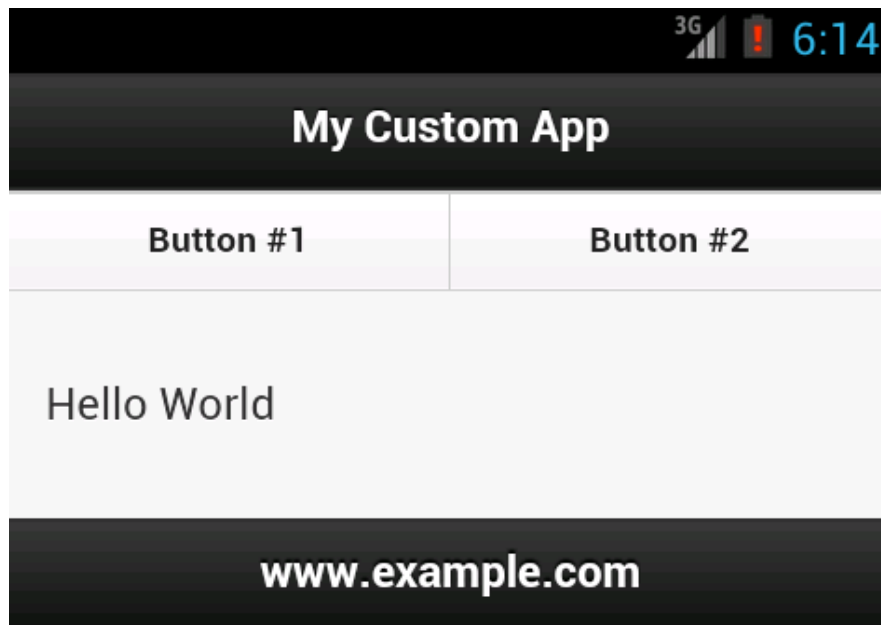


DrupalGap comes with a few core system blocks to cover the basic needs of most apps. The **title**, **main\_menu**, and **main** blocks are core blocks, and **my\_custom\_block** is a custom block we'll be creating shortly for this example.

More info on blocks: <http://www.drupalgap.org/node/83>

# Using Blocks for Display within a Theme's Regions

In this example, we have one block per region. The **title** block in the header region is used to display the page title, the **main\_menu** block in the navigation region shows our menu links, the **main** block in the content region shows the body of the page, and the **my\_custom\_block** in the footer region shows a custom URL.



# Placing Blocks in Regions on a Theme using the settings.js file

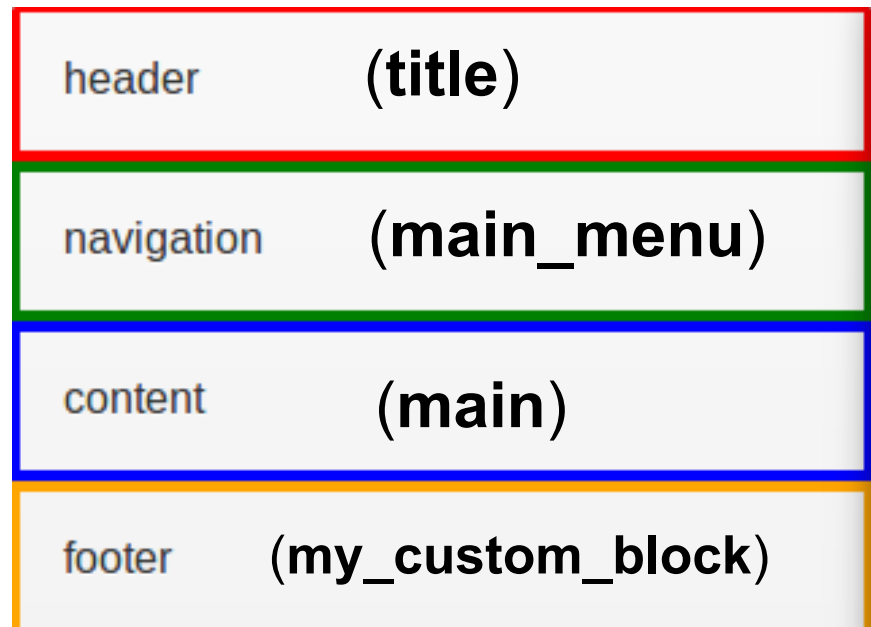
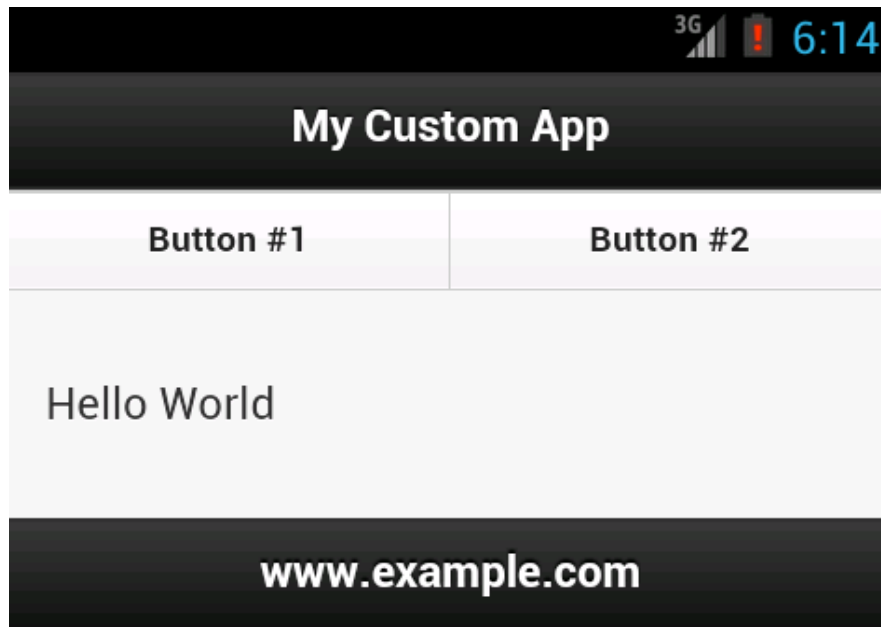
```
// Easy Street 3 Theme Blocks
```

```
drupalgap.settings.blocks.easystreet3 = {  
  'header':{ 'title':{} },  
  'navigation':{ 'main_menu':{} },  
  'content':{ 'main':{} },  
  'footer':{ 'my_custom_block':{} }  
};
```

Notice the **Block** settings uses our **Theme's** machine name, **easystreet3**. Then each **Region** is keyed by its machine name, and each **Block** is keyed by its **delta** name.

# Using Menus in DrupalGap

Let's use our custom module to create two pages that will be shown when the **main\_menu** buttons are clicked.



*Example code available on next two slides...*

```
/**
 * Implements hook_menu().
 */
function my_module_menu() {
  var items = {
    my_custom_page:{
      title:"My Custom App",
      page_callback:"my_module_custom_page"
    },
    button_one_page:{
      title:"Button One Page",
      page_callback:"my_module_button_one_page"
    },
    button_two_page:{
      title:"Button Two Page",
      page_callback:"my_module_button_two_page"
    }
  };
  return items;
}
```



```
function my_module_button_one_page() {  
  return {  
    my_text:{  
      markup:"<p>Hello Button One World</p>"  
    }  
  };  
}
```

```
function my_module_button_two_page() {  
  return {  
    my_other_text:{  
      markup:"<p>Hello Button Two World</p>"  
    }  
  };  
}
```

# Now that the Pages are created, let's Add some Links to the Main Menu...

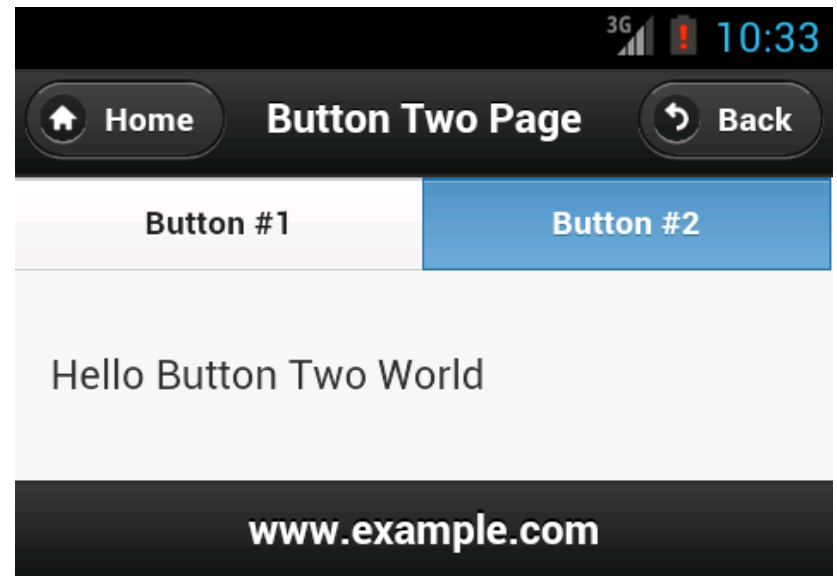
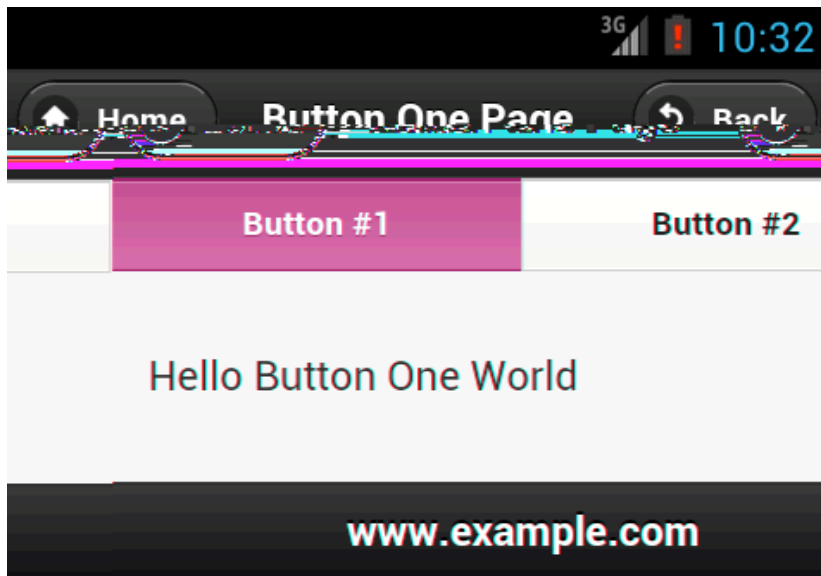
Since DrupalGap comes with a **main\_menu**, we can add two links to it using this code in the **settings.js** file:

```
// Main Menu
drupalgap.settings.menus['main_menu'] = {
  links:[
    {
      title:'Button #1',
      path:'button_one_page'
    },
    {
      title:'Button #2',
      path:'button_two_page',
    }
  ]
};
```

More info about creating menus and links: <http://www.drupalgap.org/node/85>

# Each Menu automatically has a Block created for it, following Drupal standards

Earlier we placed the main\_menu **Block** in the navigation **Region**, so if we run the App, the menu buttons can now be clicked to see the new **Pages**. All that is left is to create the custom **Block** for the footer **Region**...



More info on navigating pages: <http://drupalgap.org/node/137>

# Creating a Custom Block with hook\_block\_info() using JavaScript

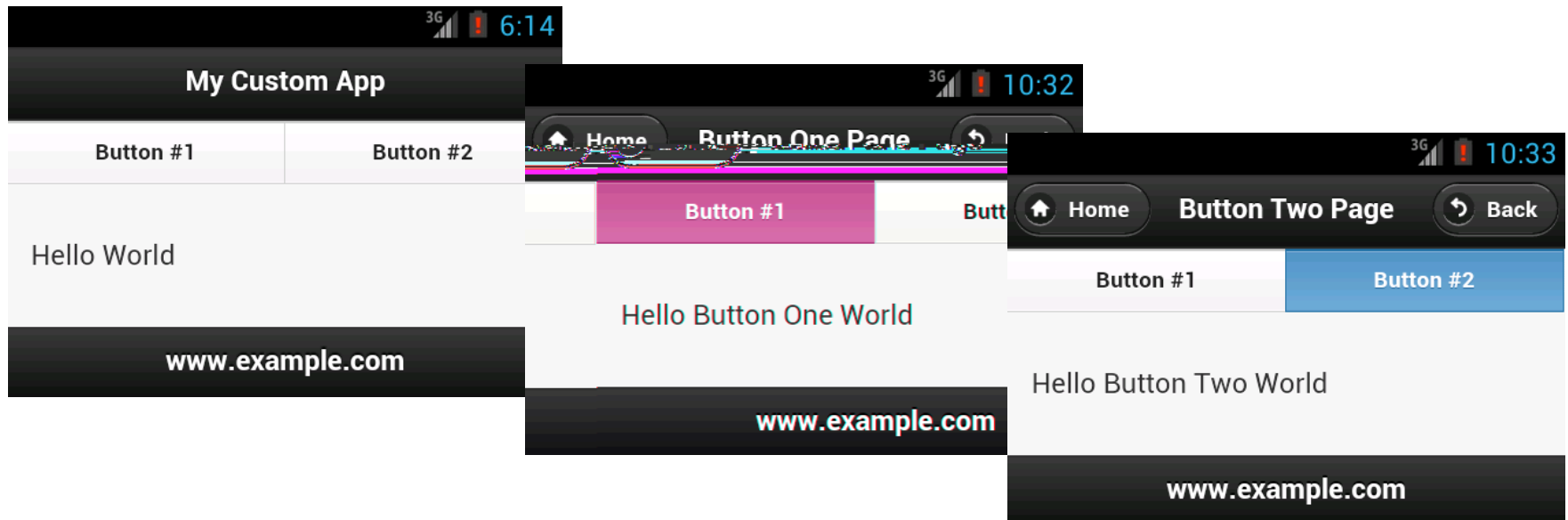
```
/**
 * Implements hook_block_info().
 */
function my_module_block_info() {
  var blocks = {
    my_custom_block:{
      delta:'my_custom_block',
      module:'my_module',
    },
  };
  return blocks;
}
```

# Setting the Block's display using hook\_block\_view() and JavaScript

```
/**
 * Implements hook_block_view().
 */
function my_module_block_view(delta) {
  var content = '';
  if (delta == 'my_custom_block') {
    content = '<h2>www.example.com</h2>';
  }
  return content;
}
```

More info about creating blocks: <http://www.drupalgap.org/node/83>

# We're now ready to view our completed pages with a menu and custom block...



By using coding techniques familiar to Drupal Developers, we now have an **App** with a **Theme**, **Regions**, **Blocks**, a **Menu** (with active links) and multiple **Pages**.

*The 'Home' and 'Back' buttons are referred to as "Region Menu Links", and are customizable. More info: <http://www.drupalgap.org/node/173>*

# Great, but what about all of my Drupal Content and Users?

DrupalGap has **Pages**, **Forms** and an **API** built in to work with Drupal's Core **Entities** and **Fields**, and any custom **Content Types** on your site.

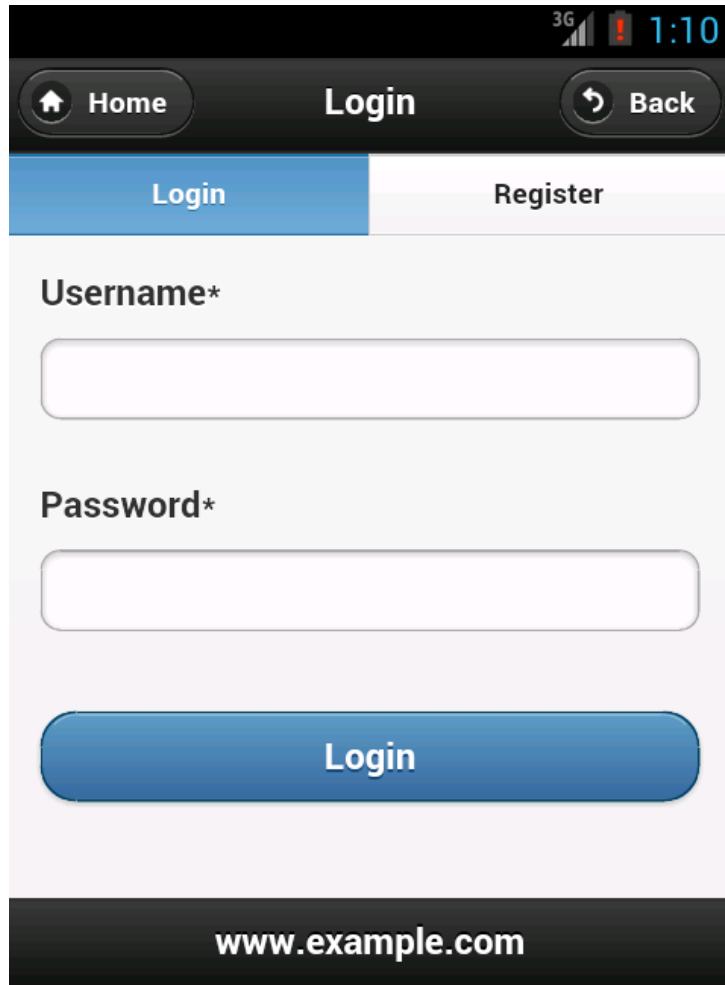
## Create, Retrieve, Update and Delete

- Nodes (*e.g. node/123, node/123/edit*)
- Users (*e.g. user/1, user/1/edit, user/login, user/register*)
- Comments
- Taxonomy Vocabularies
- Taxonomy Terms

For consistency, the core DrupalGap **page paths** (listed above) are the same as the **page paths** utilized in Drupal.

*Please note, not all core Fields are supported yet. Screenshots available on next slides...*

# User Login and Registration Pages



A mobile app screenshot of the Login page. The status bar at the top shows 3G signal, a battery icon, and the time 1:10. The app's navigation bar is black with a 'Home' button (house icon), the title 'Login', and a 'Back' button (arrow icon). Below the navigation bar is a tab bar with 'Login' (selected, blue) and 'Register' (white). The main content area is light gray and contains two text input fields: 'Username\*' and 'Password\*'. Below these fields is a large blue button labeled 'Login'. At the bottom is a black footer bar with the text 'www.example.com'.

3G 1:10

Home Login Back

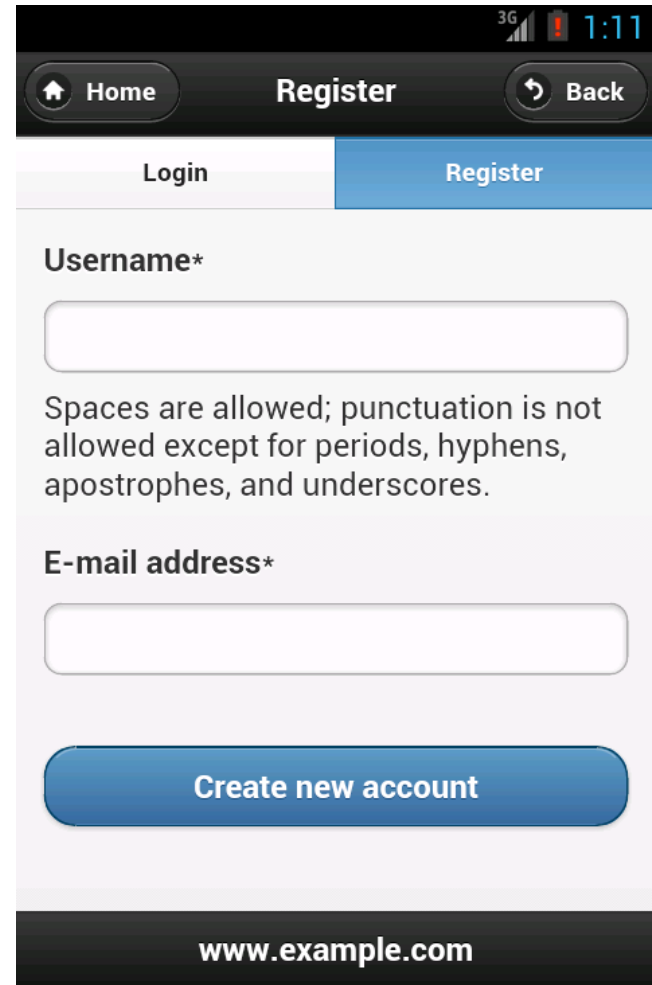
Login Register

Username\*

Password\*

Login

www.example.com



A mobile app screenshot of the Register page. The status bar at the top shows 3G signal, a battery icon, and the time 1:11. The app's navigation bar is black with a 'Home' button (house icon), the title 'Register', and a 'Back' button (arrow icon). Below the navigation bar is a tab bar with 'Login' (white) and 'Register' (selected, blue). The main content area is light gray and contains three text input fields: 'Username\*', 'E-mail address\*', and a third field for a password. Below the 'E-mail address\*' field is a large blue button labeled 'Create new account'. At the bottom is a black footer bar with the text 'www.example.com'.

3G 1:11

Home Register Back

Login Register

Username\*

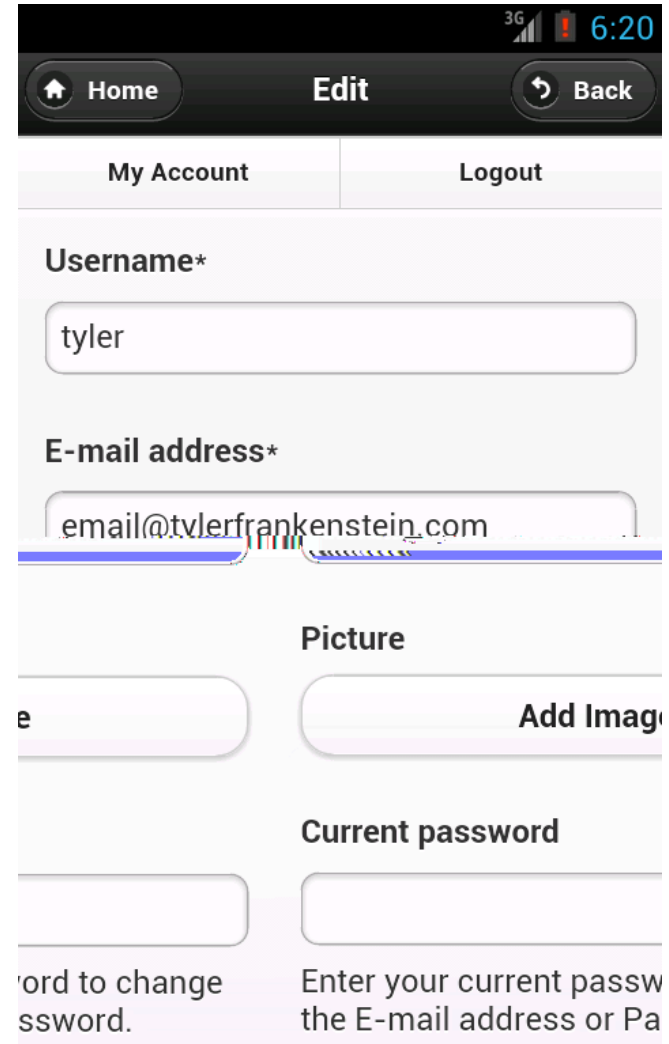
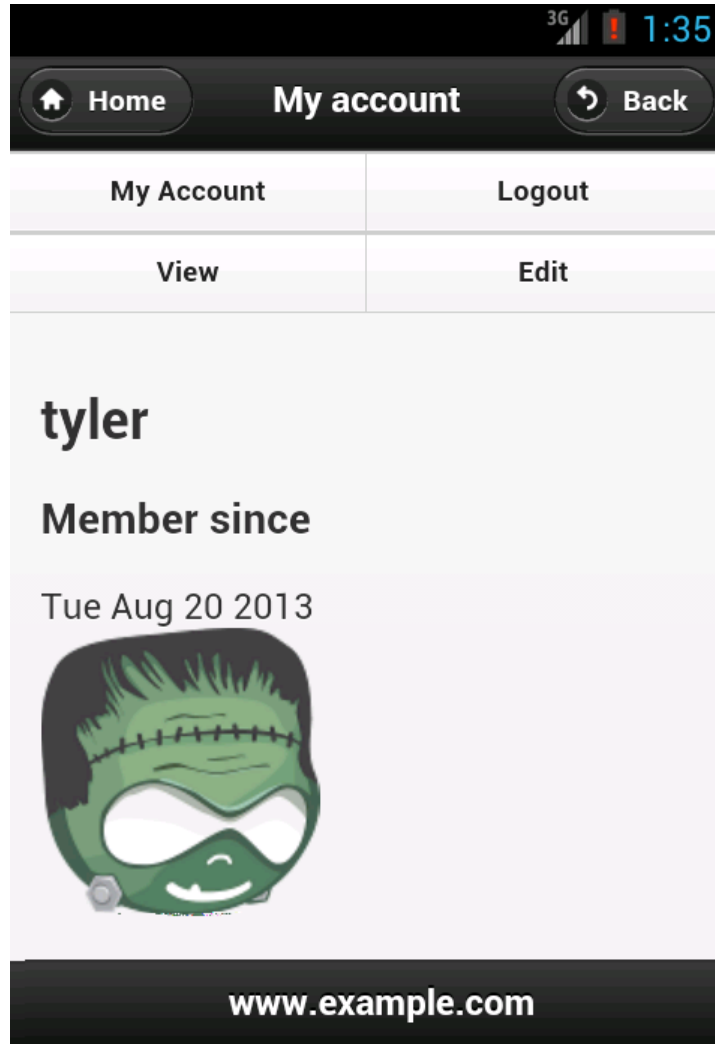
E-mail address\*

Create new account

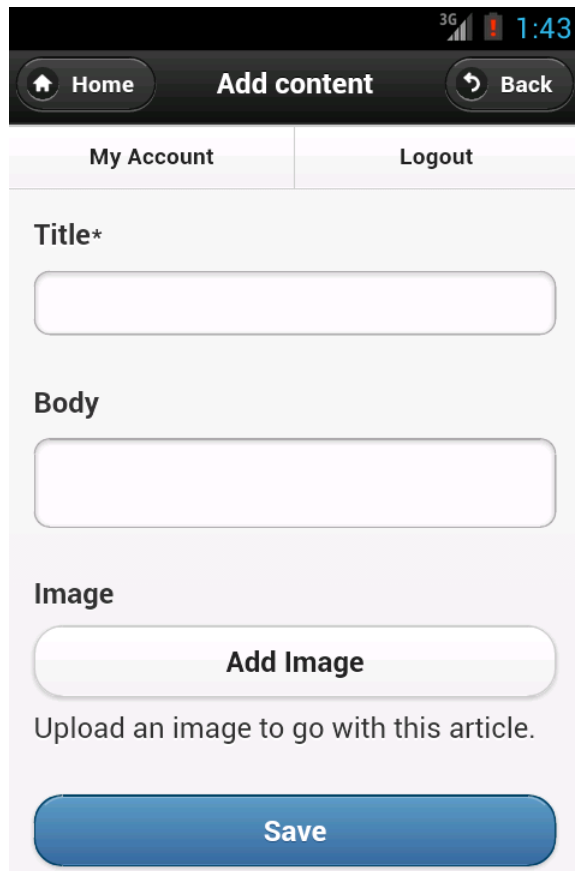
www.example.com



# View User Accounts, Profile Pictures and Edit user Accounts

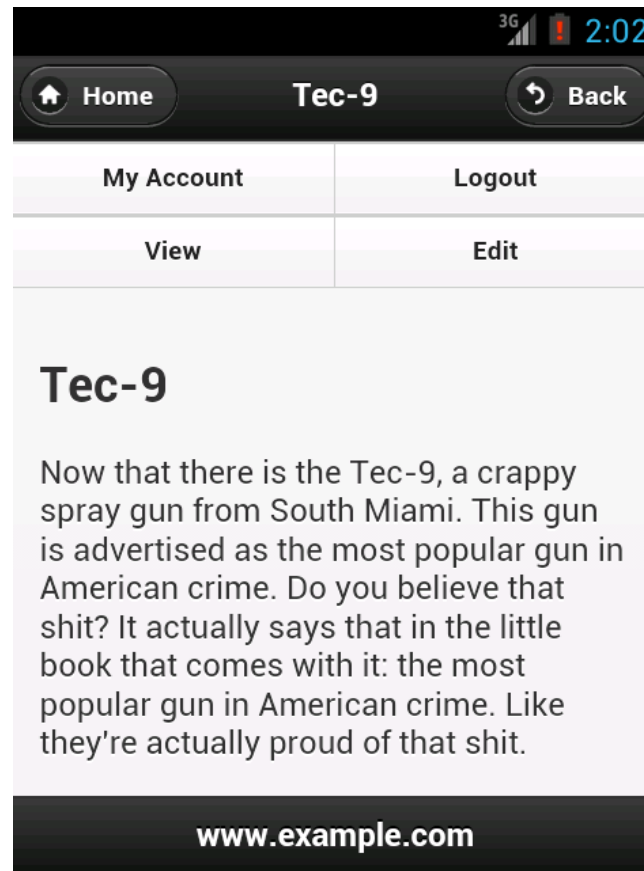


# Node Content Creation, Viewing and Editing



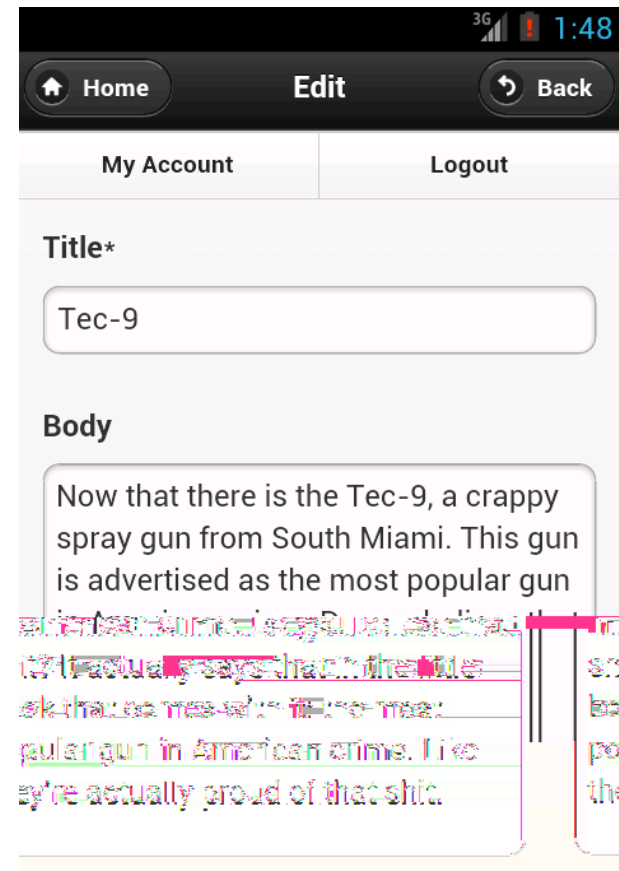
Mobile app interface for adding a new article. The top bar shows 'Home', 'Add content', and 'Back'. Below the bar are 'My Account' and 'Logout' links. The main form has three sections: 'Title\*' with a text input field, 'Body' with a larger text input field, and 'Image' with an 'Add Image' button and the text 'Upload an image to go with this article.' At the bottom is a large blue 'Save' button.

node/add/article



Mobile app interface for viewing an article titled 'Tec-9'. The top bar shows 'Home', 'Tec-9', and 'Back'. Below the bar are 'My Account' and 'Logout' links. The main content area shows the title 'Tec-9' and a paragraph of text: 'Now that there is the Tec-9, a crappy spray gun from South Miami. This gun is advertised as the most popular gun in American crime. Do you believe that shit? It actually says that in the little book that comes with it: the most popular gun in American crime. Like they're actually proud of that shit.' At the bottom is a black bar with the text 'www.example.com'.

node/123



Mobile app interface for editing an article titled 'Tec-9'. The top bar shows 'Home', 'Edit', and 'Back'. Below the bar are 'My Account' and 'Logout' links. The main form has three sections: 'Title\*' with a text input field containing 'Tec-9', 'Body' with a larger text input field containing the same paragraph of text as the view page, and an 'Add Image' button. At the bottom is a large blue 'Save' button.

node/123/edit

# Image Field Support

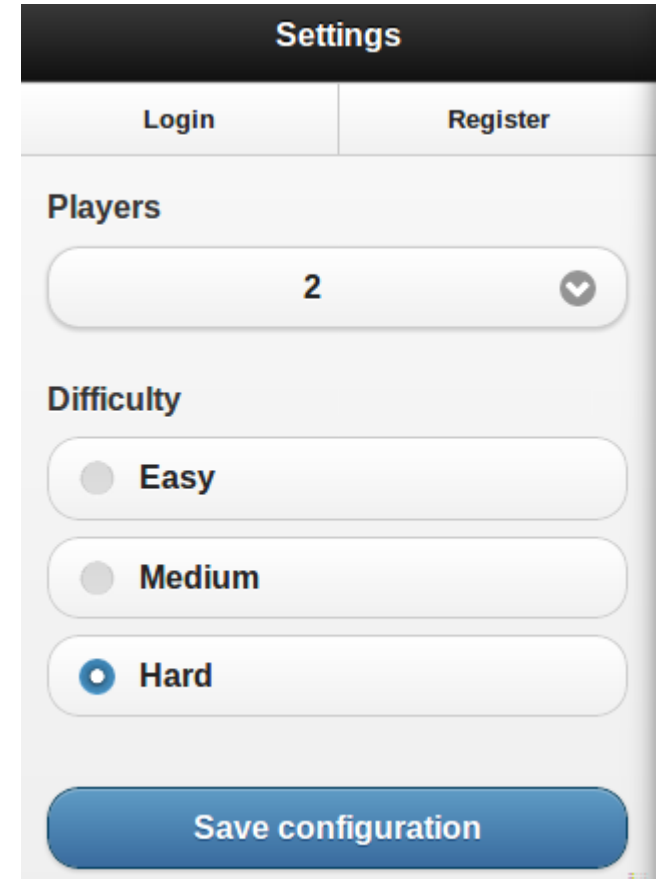
Take a **picture** with a **mobile device** and **upload** it directly to a **node** with an **image field**.



*“Actual photo of live DrupalGap development, exciting!”*

# Forms API

- Create Custom Forms
  - Checkboxes
  - Email Fields
  - Hidden Fields
  - Password Fields
  - Radio Buttons
  - Select Lists
  - Text Areas
  - Text Fields
- Form Validation
- Form Submission
- Alter Forms [ with *hook\_form\_alter()* ]



The screenshot shows a 'Settings' form with two tabs: 'Login' and 'Register'. The 'Players' section has a text field containing the number '2' and a dropdown arrow. The 'Difficulty' section has three radio buttons labeled 'Easy', 'Medium', and 'Hard', with 'Hard' being selected. At the bottom is a blue button labeled 'Save configuration'.

More info about Forms: <http://www.drupalgap.org/node/76>

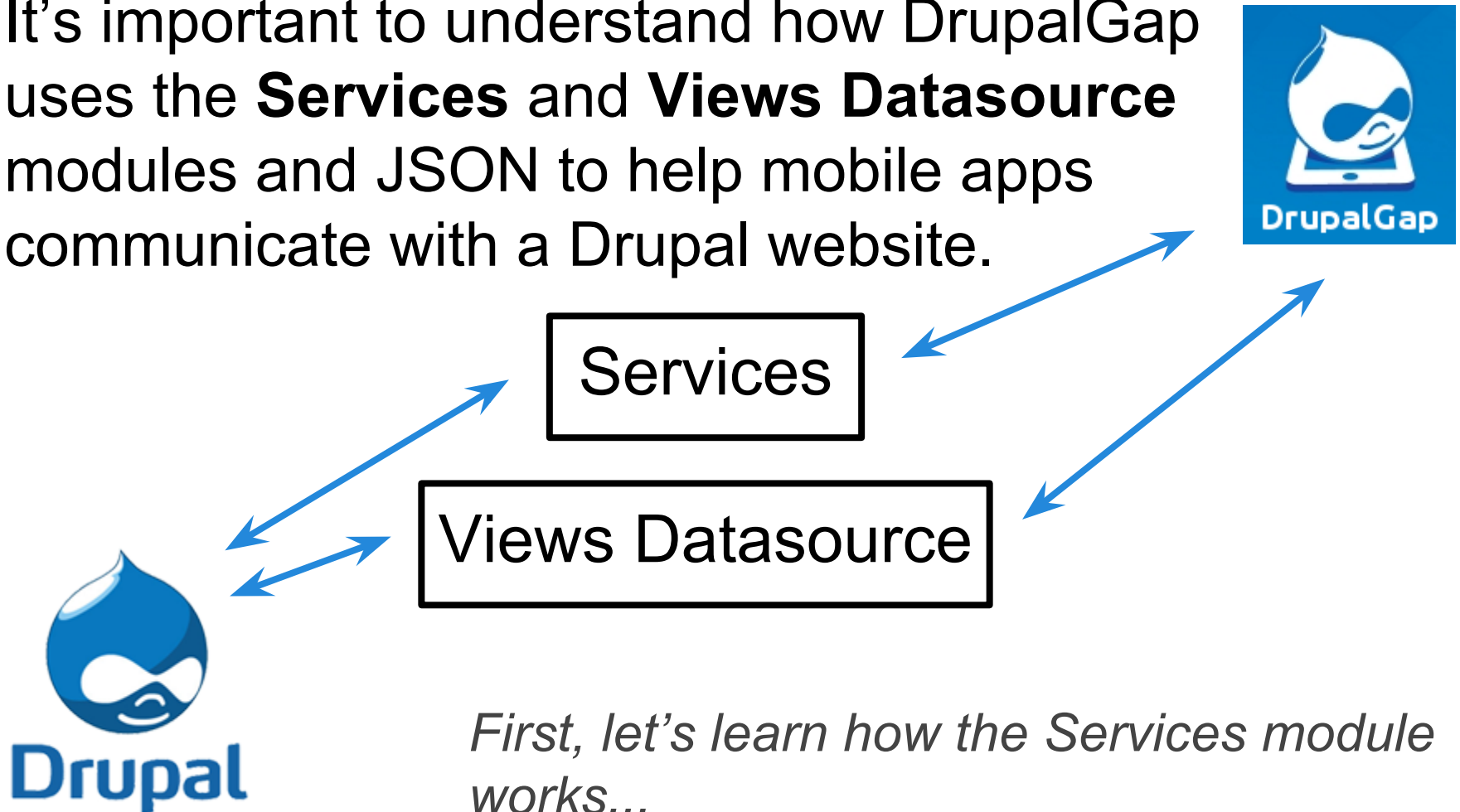
# And much more...

- Page Transitions
- Block Visibility Rules
- Working with Views Results
- Lists
- Images
- Buttons
- CSS
- RSS
- Services API
- System Settings Forms
- Page Caching
- Local Storage Entity Caching
- Performance Tweaks

*Examples and code available online at: <http://www.drupalgap.org>*

# DrupalGap does much of the work for us...

It's important to understand how DrupalGap uses the **Services** and **Views Datasource** modules and JSON to help mobile apps communicate with a Drupal website.



*First, let's learn how the Services module works...*

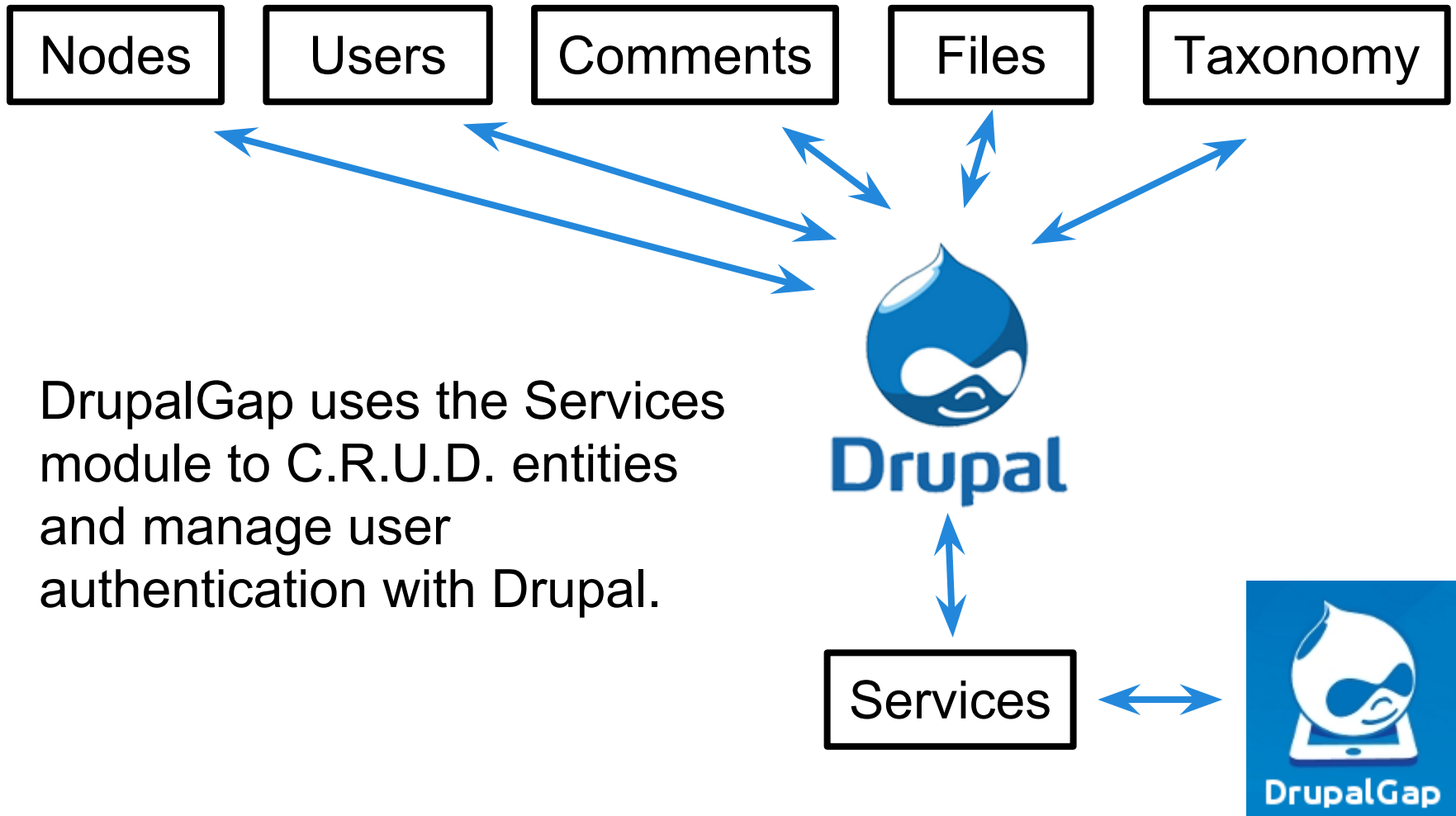
# How does the Services module work?

The Services module provides URLs that can be used by a mobile application to C.R.U.D. entities on a Drupal website.

**C.R.U.D** (*create, retrieve, update, delete*)

- Nodes
- Users (*login, logout, registration, sessions*)
- Comments
- Files
- Taxonomy

# How does DrupalGap use the Services module?





# An Example Service Call

## **Retrieve a Node** (HTTP GET)

`http://example.com/drupalgap/node/123.json`

```
{  
  nid:123,  
  title:"Hello World",  
  uid:1,  
  type:"article"  
}
```

*Note, in this example our Service endpoint name is 'drupalgap' and it shows just a small sample of the JSON data returned for a node.*

# Another Example Service Call

## **Register a User** (HTTP POST)

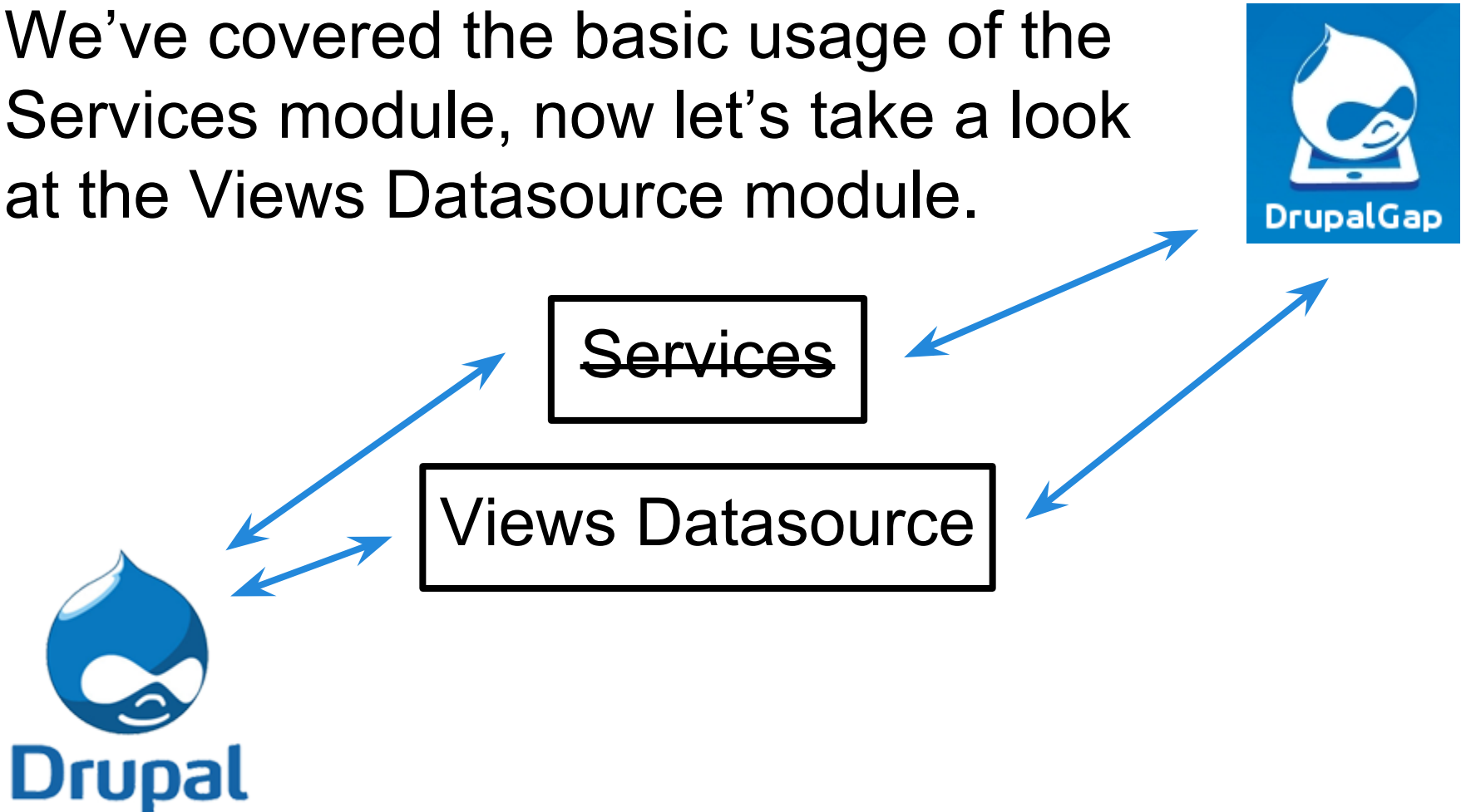
http://example.com/drupalgap/user/register.json?  
**name=bob&mail=bob@bob.com**

```
{  
  uid:456,  
  uri:"http://www.example.com/drupalgap/user/456.json"  
}
```

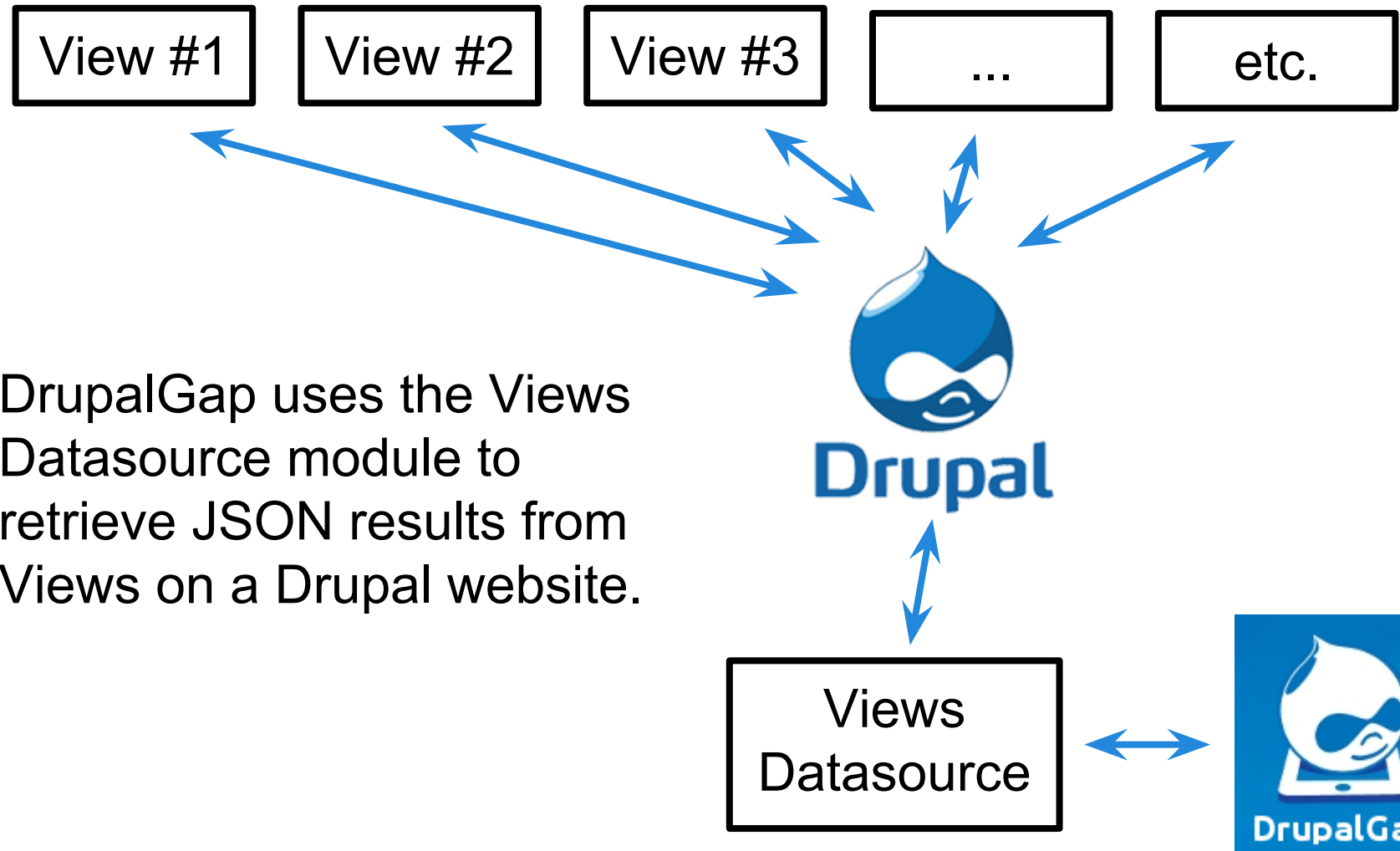
*Note, the mail value would need to be URL encoded as bob%40bob.com before being posted.*

# What about Views Datasource?

We've covered the basic usage of the Services module, now let's take a look at the Views Datasource module.



# How does DrupalGap use the Views Datasource module?



# How does the Views Datasource module work? An example...

Let's first imagine a simple page created by a View in Drupal:

## URL

- <http://www.example.com/articles>

## Format

- Table

## Fields

- Node ID (hidden)
- Title
- Post date

## Filter

- Only content of type **Article**

## Sort Criteria

- Post date (descending)

# Views Datasource example continued...

The simple page created by the Drupal View would look something like this in a web browser:

<http://www.example.com/articles>

## Articles

Title	Post date
<a href="#">Monday Press</a>	Monday, September 30, 2013 - 18:12
<a href="#">Sunday Post</a>	Sunday, September 29, 2013 - 18:11
<a href="#">Saturday Message</a>	Saturday, September 28, 2013 - 18:10

This works great for generating the HTML to view the page, but we would want this data in JSON format for a mobile app.

# Views Datasource example continued...

By adding a new '**Page display**' to our View, setting its URL path to '**articles/json**' and its format to '**JSON data document**', we can get that same data in JSON format.

<http://www.example.com/articles/json>

Now if our mobile application were to use an **HTTP GET** on the new Page URL, all of the Fields specified in the View will be available as JSON.

*(example JSON data available on next slide)*

```
{
  "nodes" : [
    {
      "node" : {
        "title" : "Monday Press",
        "nid" : "160",
        "created" : "Monday, September 30, 2013 - 18:12"
      }
    },
    {
      "node" : {
        "title" : "Sunday Post",
        "nid" : "159",
        "created" : "Sunday, September 29, 2013 - 18:11"
      }
    },
    {
      "node" : {
        "title" : "Saturday Message",
        "nid" : "158",
        "created" : "Saturday, September 28, 2013 - 18:10"
      }
    }
  ]
}
```



# Another Views Datasource example...

Imagine another simple page created by a View in Drupal:

## URL

- <http://www.example.com/latest-users>

## Format

- JSON data document

## Fields

- Name
- User ID

## Filter

- Only **Users** who are **Active**

*(example JSON data available on next slide)*

```
{  
  "users": [  
    {  
      "user": {  
        "name": "bob",  
        "uid": "1"  
      }  
    },  
    {  
      "user": {  
        "name": "betty",  
        "uid": "2"  
      }  
    }  
  ]  
}
```

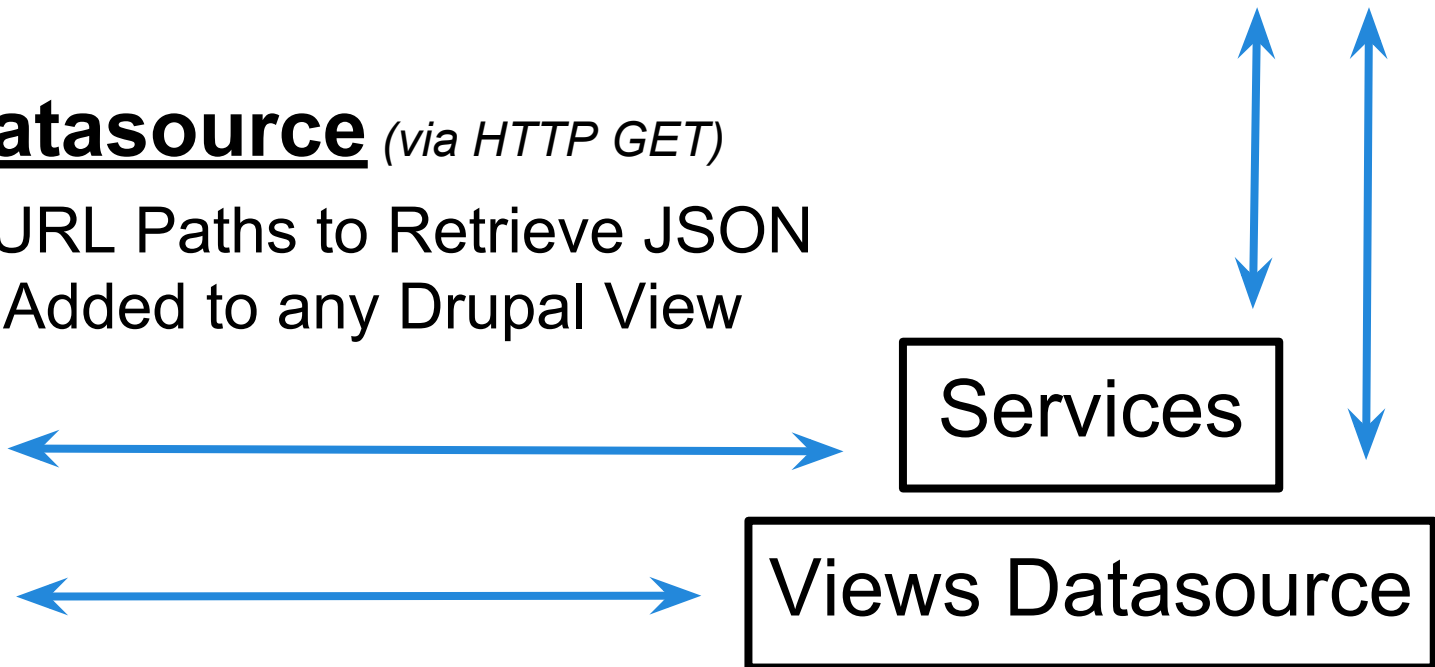
# An overview of how DrupalGap uses the Services and Views Datasource modules

## Services *(via HTTP methods)*

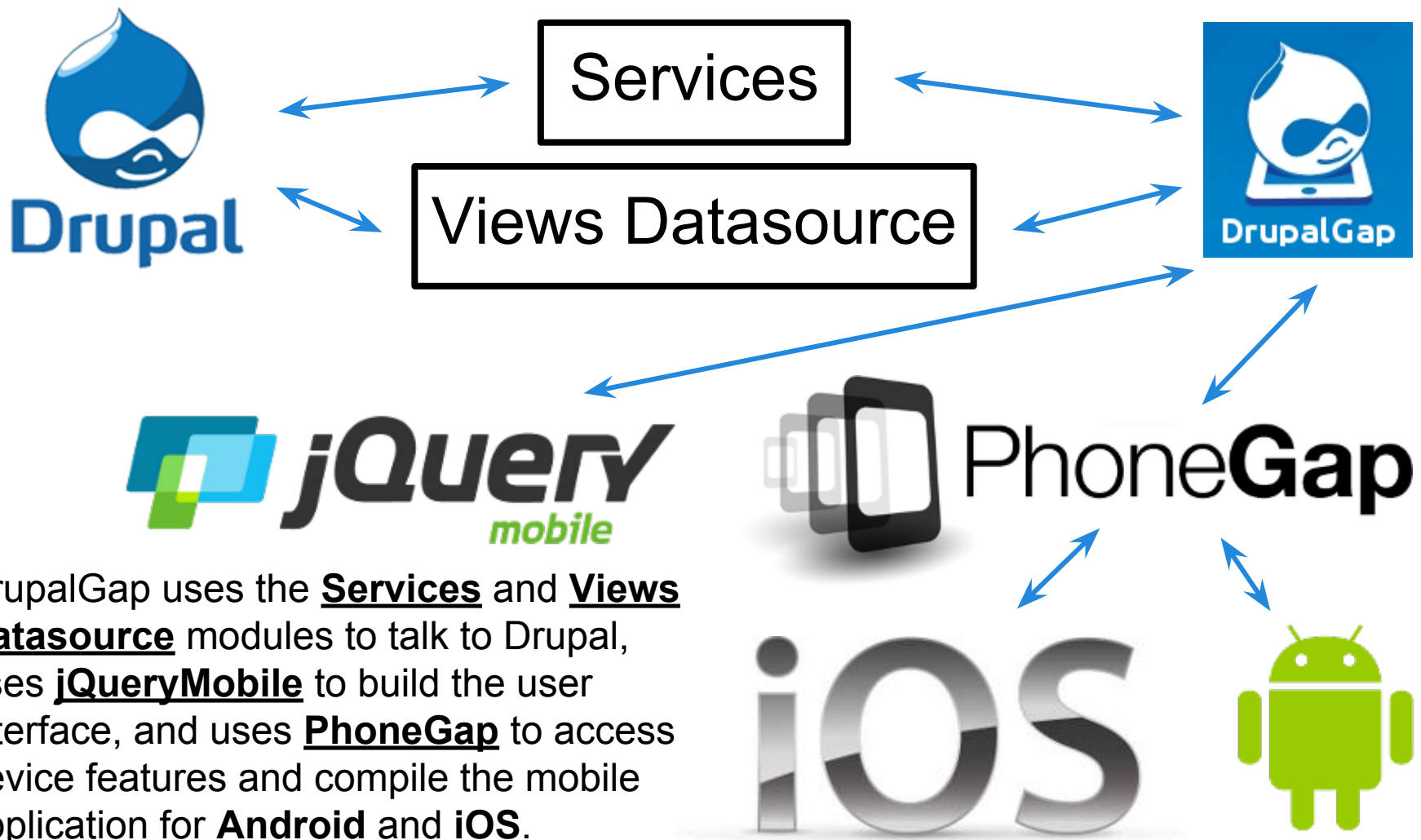
- **C.R.U.D.** *(create, retrieve, update, delete)*
  - **Entities** *(nodes, users, comments, files, taxonomy)*
- **User** *(login, logout, registration, sessions)*

## Views Datasource *(via HTTP GET)*

- Create URL Paths to Retrieve JSON
- Can be Added to any Drupal View



# Putting it all together...



DrupalGap uses the **Services** and **Views Datasource** modules to talk to Drupal, uses **jQueryMobile** to build the user interface, and uses **PhoneGap** to access device features and compile the mobile application for **Android** and **iOS**.

# An overview of the DrupalGap Mobile Application Development Kit



## Utilizes Drupal Concepts for Content and UI

- Themes
- Regions
- Blocks
- Menus
- Pages

## Utilizes Drupal Customization Techniques

- Modules
- Hooks
- Themes
- Templates

## JSON API

- Services (*C.R.U.D. of Drupal Entities*)
- Views Datasource (*Retrieve Views Results*)

# DrupalGap's API and Open Source...

## **JavaScript Functions** - <http://api.drupalgap.org>

- `node_load(123);`
- `user_load(456);`
- `l('My Link', 'my_link_url_path');`
- `arg(1);`
- `user_access('access content');`
- *etc...*

## **Open Source**

- GitHub (6 contributors, 46 forks, 73 stars)\*
- 400+ Drupal Installs\*
- Contributed Modules and Sandboxes

*\* As of October 11th, 2013...*

# Getting Started with DrupalGap



<http://www.drupalgap.org>

- Hello World
- Getting Started Guide
- Example Code
- Troubleshooting Topics
- Contributed Modules and Sandboxes

<http://api.drupalgap.org>



*“Thank you, and happy coding!”*

- Tyler Frankenstein