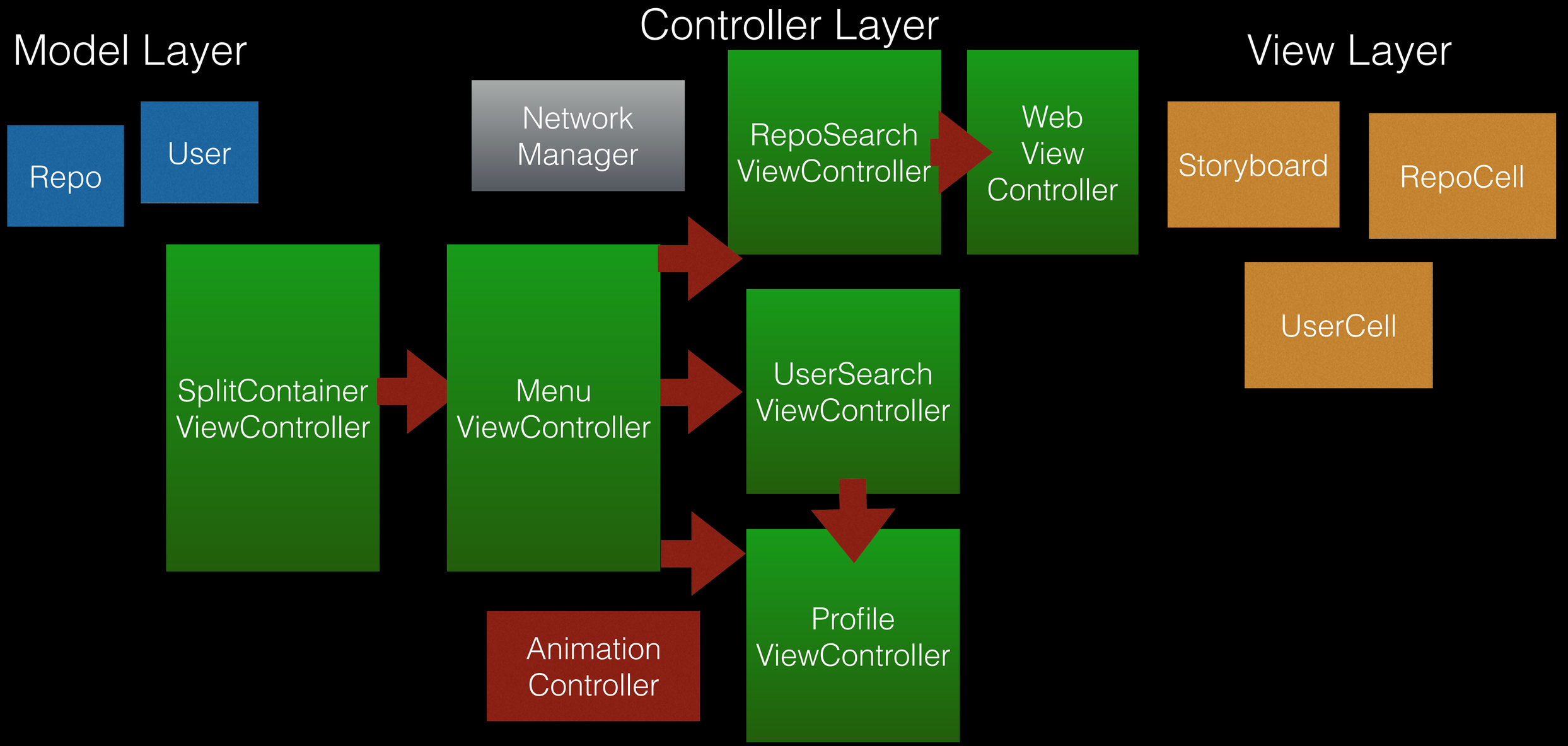
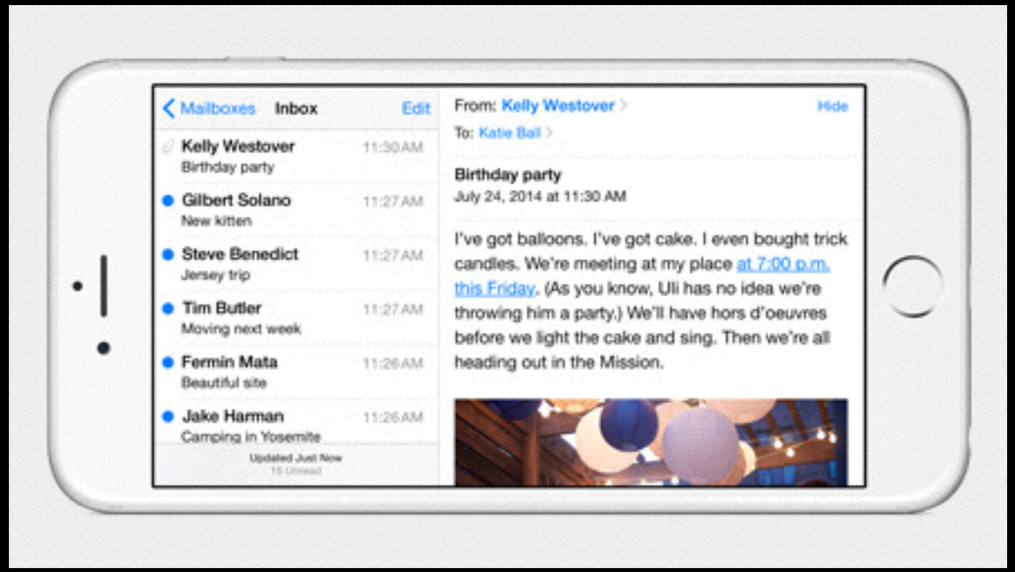
iOS Dev Accelerator Week3 Day1

- UISplitViewController
- HTTP
- NSURLSession
- Git (if we have time)
- Mobile Monday: Android views and grid view

The MVC layout of our Week 3 App





UISplitViewController

UISplitViewController

- “UISplitViewController is a container view controller that manages the presentation of two side-by-side view controllers.”
- Typically the left master side will display a list, and the right side presents details of the selected item.
- the UISplitViewController object does not have a user facing interface in itself, it just manages the other view controllers.
- changing the primary and detail view controllers is as simple as calling the appropriate methods:
  - showViewController:sender: or showDetailViewController:sender:
- Some of its functionality is achieved with a delegate

Child View Controllers

- UISplitViewController has a property called viewControllers, which is an array of the children view controllers it is managing.
- When the split view controller is expanded, this array has two view controllers.
- When the split view controller is collapsed the property only contains one view controller.

Demo

Collapsed Split View Controller

- When a split view controller is ‘collapsed’, it means there is now only one child view controller it is managing.
- **Collapsing happens when the split view controller is in a horizontally compact environment (ie iPhone except iPhone 6 plus landscape!)**
- When not in a horizontal compact environment, the split view controller is in its natural expanded state.

Expanded Split View Controller

- So when your split view controller is expanded, it tries to display the master view controller and the detail controller at the same time.
- It figures out its layout based on its displayMode.
- UISplitViewControllerDisplayMode is an enum with 4 values:
  - Automatic - its up to the split view controller to decide the most appropriate display mode
  - PrimaryHidden - the primary view controller is hidden
  - AllVisible - the primary and secondary view controllers are displayed side-by-side
  - PrimaryOverlay - the primary view controller is layered on top of the secondary view controller, leaving the secondary partially visible

Changing the view controller

- New with iOS8, there are two separate methods added to UIViewController for changing the view controllers in a split view controller:
  - showViewController:sender:
  - showDetailViewController:sender:
- regular show just acts like a regular push segue that pushes a VC onto the primary view controller's navigation stack. If there is no nav controller, it does a modal present.
- showDetail bubbles up the view controller chain, and if it finds a split view controller, it pushes a VC as the secondary VC if the split view is expanded, and it pushes it onto the primary VC if the split view is in collapsed state. And again, if it gets all the way up and doesn't find a navigation controller, it does a modal present.

Delegate

- UISplitViewControllerDelegate protocol defines methods that allow you to manage changes to a split view controller interface.
- Most of the methods allow you to respond to change in the display mode and user interface orientation.
- You also get notified of when a split view controller collapses and expands, and even when a new view controller is added to the interface.

Delegate on collapse

- For our Github app, we are going to implement one delegate method:
- splitViewController:collapseSecondaryViewController:
- This method is fired when the split view is going to collapse.
- It returns a bool, returning true tells the split view to throw away the secondary view controller. Returning false tells the split view to take the secondary view controller and push it onto the primary's navigation stack.
- On launch, we want this to return true so the user is taken to the main menu at first. Otherwise they would be staring at one of our detail view controllers at first and might not realize how much more functionality the app contains.

Demo

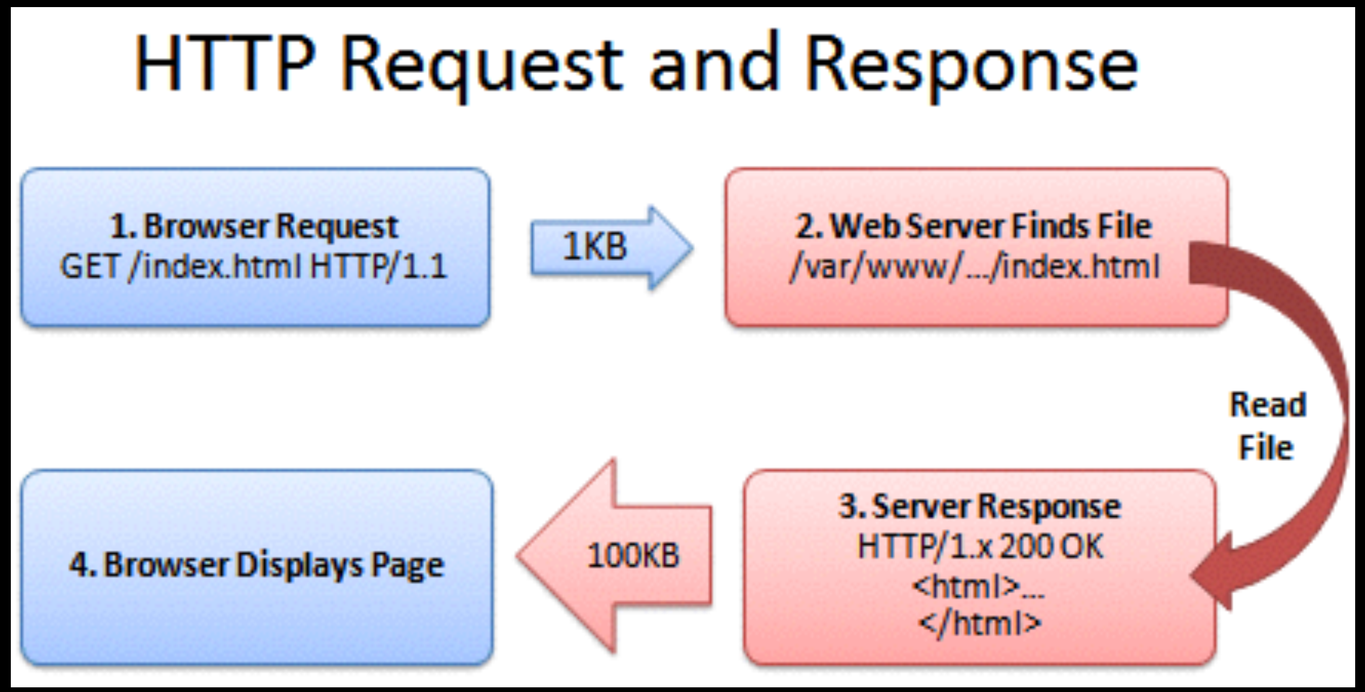
HTTP

HTTP (Hyper Text Transfer Protocol)

- HTTP is a protocol that the internet uses to format and transmit resources (pages,files, data) on the web.
- A resource is anything that can be identified by a URL. It is the R in URL!
- It defines how web servers and clients should respond to various commands.
- When entering a URL into your browser, you are just sending an HTTP command to a web server to retrieve a resource.
- So your browser is considered an *HTTP Client*, and it sends requests to an *HTTP Server*.
- The default port for HTTP servers to listen on is port 80, but they can use any port.
- It is a stateless protocol, because each request does not know about any of the previous requests.

HTTP (Hyper Text Transfer Protocol)

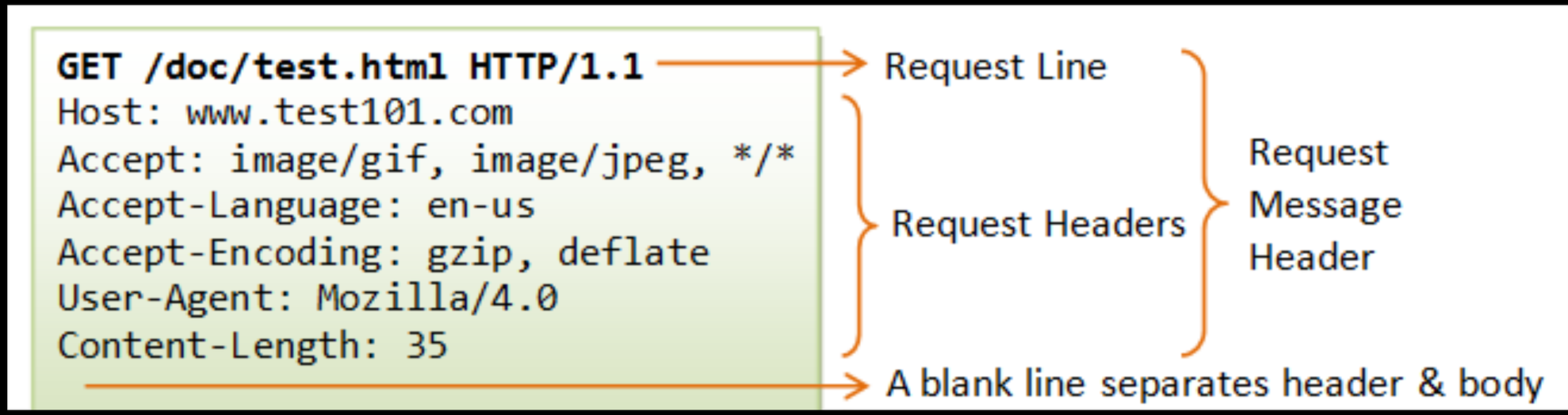
- HTTP uses a client-server model.
- A client sends a request to a server, and then the server returns a response message, which most often will contain the resource that was requested.



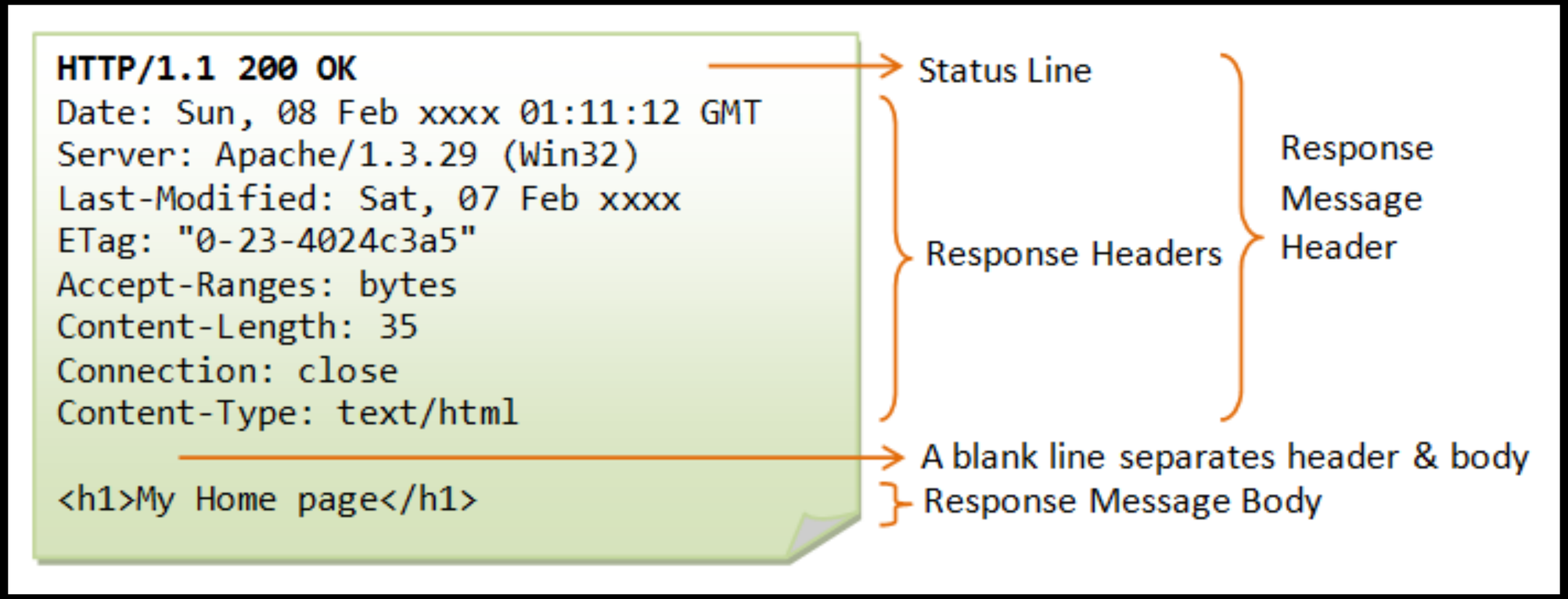
Request and Response Format

- The format of a request and response are very similar.
- They are considered ‘English-oriented’ and human readable.
- Both start off with an initial line.The initial line is where the main differences are between requests and responses.
- After the initial line, there can be zero or more header lines.
- After the header line is a blank line
- And finally you have the optional message body.

Request Example



Response Example



Request Initial Line

- A request line has three parts and they are **separated by spaces**.
- The first part is the method name (more on this in a bit)
- The second part is the local path of the request resource
- and the last part is the version of HTTP being used
- Example:

GET /NFL/seahawks/tickets.html HTTP/1.1

Response Initial Line, aka the Status Line

- A status line has three parts and they are **separated by spaces**.
- The first part is the version of HTTP being used
- The second part is a response status code
- and the last part is an english reason phrase describing the status
- Example:

HTTP/1.1 404 Not Found

HTTP Methods (Verbs)

- GET - Most common HTTP method. Retrieves whatever resource is at the URL location. Your browser history is just a history of all the GET requested you have made.
- POST - Method used to request that the server accept data enclosed in the request. When you tweet, you are using POST.
- DELETE - Used to delete a resource.

HTTP Status Codes

- For the most part only a few status codes that mobile apps need to check for
- 200 OK - standard response for a successful HTTP request
- 400 Bad - bad request, most likely syntax error
- 401 Unauthorized - authentication was required but not provided or incorrect in request
- 403 Forbidden - Request was valid, but the server refuses to respond.
- 404 Not found - The requested resource was not found
- 429 Too Many Requests - rate limited
- 5xx Server Error - not your app’s fault!

Header lines

- Header lines are one line per header and they take the format of:
  - “Header-Name: value”
- The header name is not case-sensitive
- You can have as many spaces or tabs between the : and the value
- Header lines that begin with space or tab are a part of the last header line for easy multi-line reading.

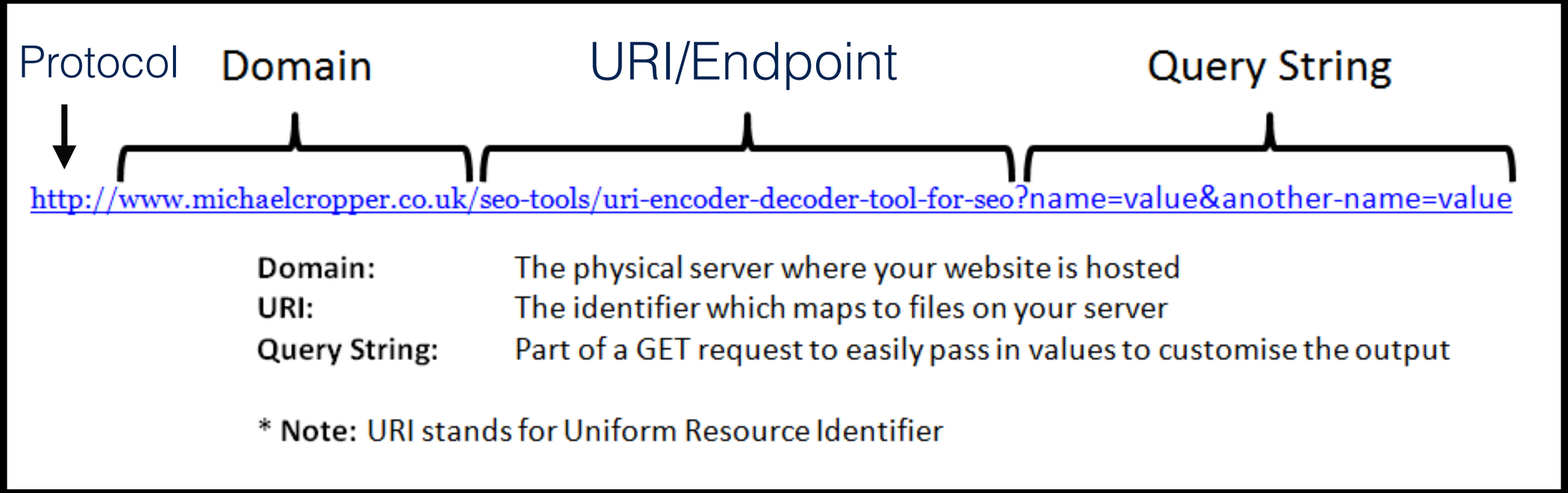
Specific Headers

- Some requests require certain header fields. Good API’s will tell you exactly which header fields you need. A lot of them won't :(
- Content-Type header field is used to specify the nature of data in the body of the request.
- Authorization header field can be used to pass credentials for protected resources. Usually an OAuth token.

The Message Body

- Any HTTP message my have a body after the header lines.
- In a request, this is where the appropriate data or files are placed in a POST.
- In a response, this is where the resource is that the request was requesting.
- Whenever this is a body, Content-Type and Content-Length are usually included in the header lines.

URL



Domain: The physical server where your website is hosted
URI: The identifier which maps to files on your server
Query String: Part of a GET request to easily pass in values to customise the output

* **Note:** URI stands for Uniform Resource Identifier

HTTP GET Example

- Lets say I wanted to get the box score for a specific mariners game. Heres the URL I would use for my GET request:
  - http://scores.espn.go.com/mlb/boxscore?gameId=340718103
- the domain is http://scores.espn.go.com
- the endpoint is mlb/boxscore
- the query string is gameId=340718103. The beginning of the query string is marked by a question mark. You can add multiple parameters by appending them with a &
- Demo!

Foundation Network Objects

- NSData - an object oriented wrapper for byte buffers (binary data).
- NSData has a number of methods to create data objects from raw bytes.
- var path = “/u/smith/myFile.txt”
- var myData = NSData(dataWithContentsOfFile:path)
- Can also go the other way:
- var image = person.Image
- var imgData = UIImagePNGRepresentation(image)

Foundation Network Objects

- NSURL : object that represents a URL that can contain the location for a resource on a remote server or a path of a file on disk.
- Can examine the URL scheme (http), host (www.example.com), path (/scripts), and query string (name=value) via properties on the NSURL object.
- Instantiated with a string.
- NSURLRequest : object for a URL Request.
- Instantiate with an NSURL object.
- Has many properties for specifying the attributes of the HTTP request (http method, body, header fields), if you need to set these things use NSMutableURLRequest

Demo

APIs

- Now that you Learned The Internet, lets look at how our iOS apps can communicate with web apps/sites
- API, or Application programming Interface, is way for parts of software to interface with other software.
- Web APIs, what you will be working with, are defined as a set of HTTP requests and response messages usually in JSON or occasionally in XML.
- Most apps on your iPhone are just clients for a web api (Facebook, Twitter, Instagram, Spotify, etc)

REST APIs

- REST is an acronym for **RE**presentational **S**tate **T**ransfer.
- You will see the term REST API a lot during your job search. It's a bit of a buzz word now, but it is essentially an API that follows certain constraints:
- Uniform Interface: Resourced based Endpoints that are consistent
- Stateless: The required state to handle the request is all contained within in the request itself. The server doesn't have to keep track of communication histories. Rate limiting is an example of stateful.
- Cacheable: Responses must defines themselves as cacheable or not
- Client-Server: Client and Server concerns are separated
- Layered System: Client doesn't know(or care) if they are connected to the main server or a load balancing server.
- Code on Demand (optional): Servers can temporarily extend or customize functionality of a client by transferring logic to them.

Web API Client workflow

1. Clients makes a request to the server at a specific endpoint
2. Server receives request, does some server magic, and then sends back response
3. Client checks the response http status code
4. If its 200 (everything is good) client parses the JSON response into model objects
5. client updates UI and state with newly acquired model objects

NSURLSession

NSURLSession

- The NSURLSession class and related classes provide an API for making HTTP requests.
- NSURLSession is highly asynchronous.
- NSURLSession has two ways to use it, with completion handlers (closures) or delegation.
- We will focus on the closure way for now.

NSURLSession Setup

- NSURLSession is initialized with a NSURLSessionConfiguration.
- NSURLSessionConfiguration has 3 types:
    1. Default session - disk based cache
    2. Ephemeral session - no disk based caching, everything in memory only
    3. Background session - similar to default, except a separate process handles all data transfers
- Configurations have many properties for customization. For example you can set the maximum number of connections per host, timeout intervals, cellular access or wifi only, and http header fields.

NSURLSession Setup

```
  var configuration =
NSURLSessionConfiguration.ephemeralSessionConfigur
ation()

    self.urlSession = NSURLSession(configuration:
configuration)
```

NSURLSession

- All http requests made with NSURLSession are considered ‘Tasks’. Think of them as little minions for your session (or don’t).
- A task must run on a session (NSURLSession class)
- Three types of tasks:
    1. Data tasks - receive and send data using NSData objects in memory
    2. Upload tasks - send files w/ background support
    3. Download tasks - retrieve files w/ background support

NSURLSession Data Task

```
var request = NSMutableURLRequest(URL: NSURL(string: "https://api.github.com/search/
repositories?q=\(string)"))
        request.HTTPMethod = "GET"

        let repoDataTask = self.urlSession.dataTaskWithRequest(request, completionHandler:
{(data, response, error) in

            if error {
                //do something for general error
            }
          else {
          //do switch statement on response code and do something with the data
          }
            })
        repoDataTask.resume()
```

Demo

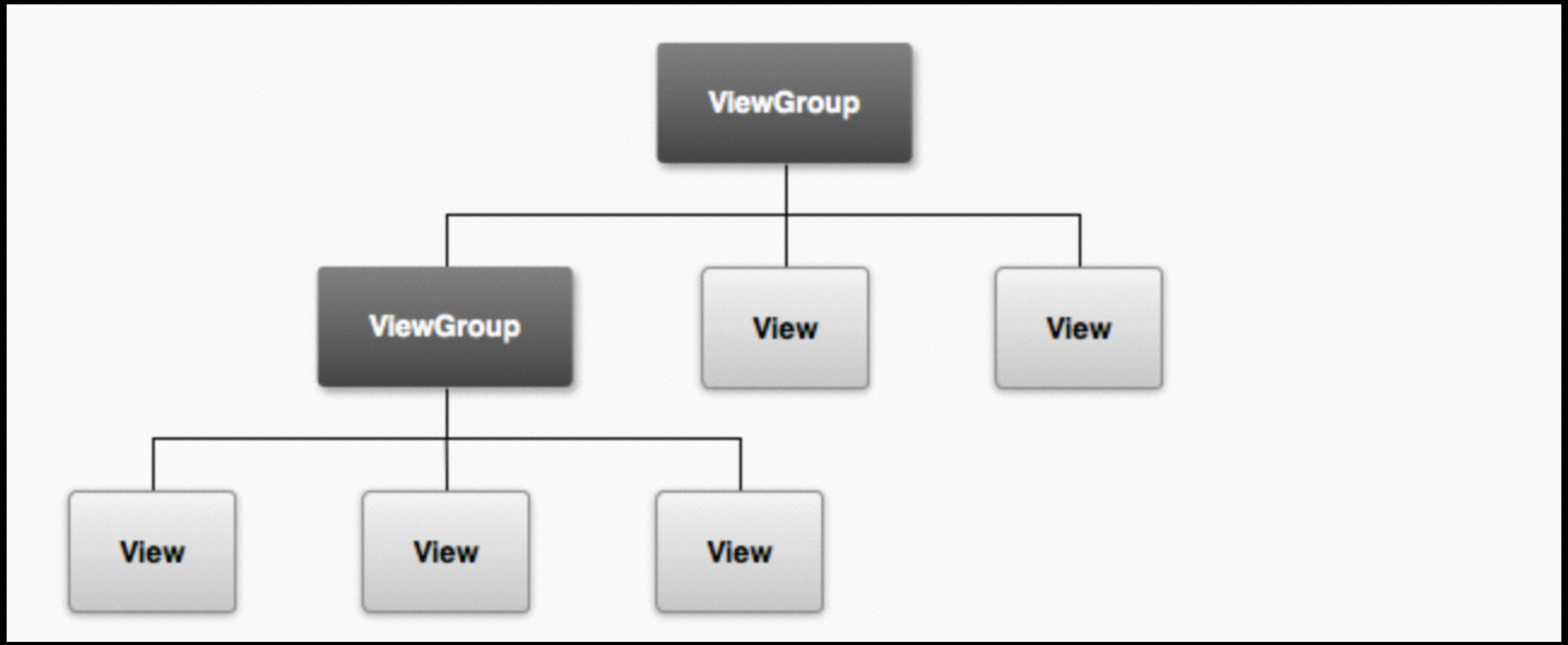
Mobile Monday

Android Interface and CollectionView

Android views

- All user interface elements in android are built with View or ViewGroup Objects.
- A View object is an object that can draw something on screen and that can be interacted with.
- A ViewGroup is an object that holds other Views and ViewGroups and defines the layout of the interface.

Android views



Android Interface Layout

- Like iOS, you can create all your views programmatically in Android, but it recommended against doing so.
- Instead, Google recommends you use XML to layout your interface:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I am a Button" />
</LinearLayout>
```

Loading the XML

- You will essentially define an XML for each Activity (aka view controller) in your android app.
- In that activity's onCreate() - which is just like viewDidLoad - you call setContentView() on your Activity and pass in a reference to your XML:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Android Grid View

- Used for the same purpose as a collection view in iOS (grid based collection)
- Grid View is a ViewGroup.
- Uses a ListAdapter, just like the List View, to display items.
- Can be placed on screen via XML:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Android Grid View

- Used for the same purpose as a collection view in iOS (grid based collection)
- Grid View is a ViewGroup.
- Uses a ListAdapter, just like the List View, to display items.
- Can be placed on screen via XML:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Android Grid View

Below is an Activity that displays the Grid View from the previous XML example

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int position, long id)
            Toast.makeText(HelloGridView.this, "" + position, Toast.LENGTH_SHORT).show
        }
    });
}
```