Integrating a RESTful Web Framework with Your iOS App

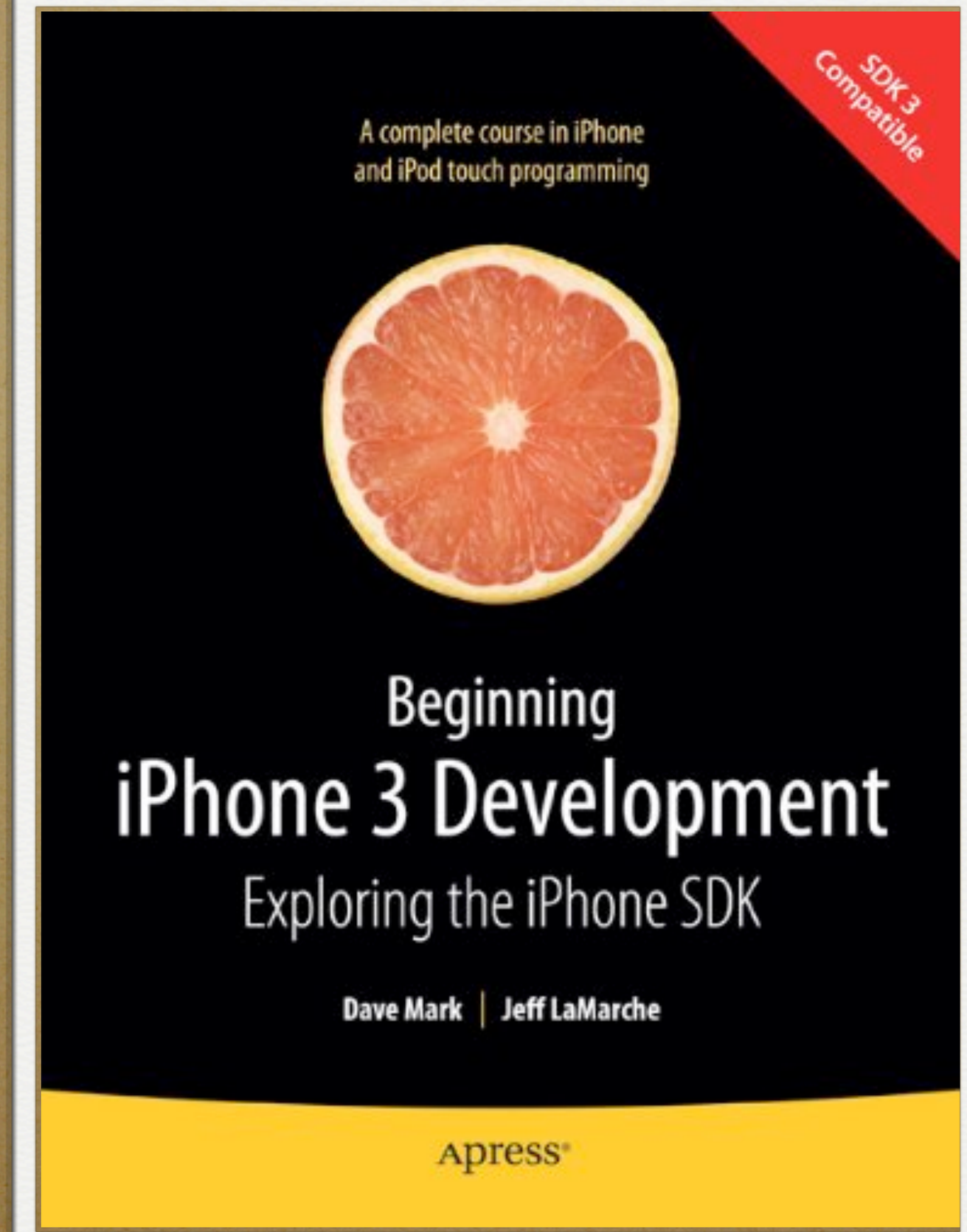
Or how I learned to stop worrying, love Core Data, and build an iOS library.

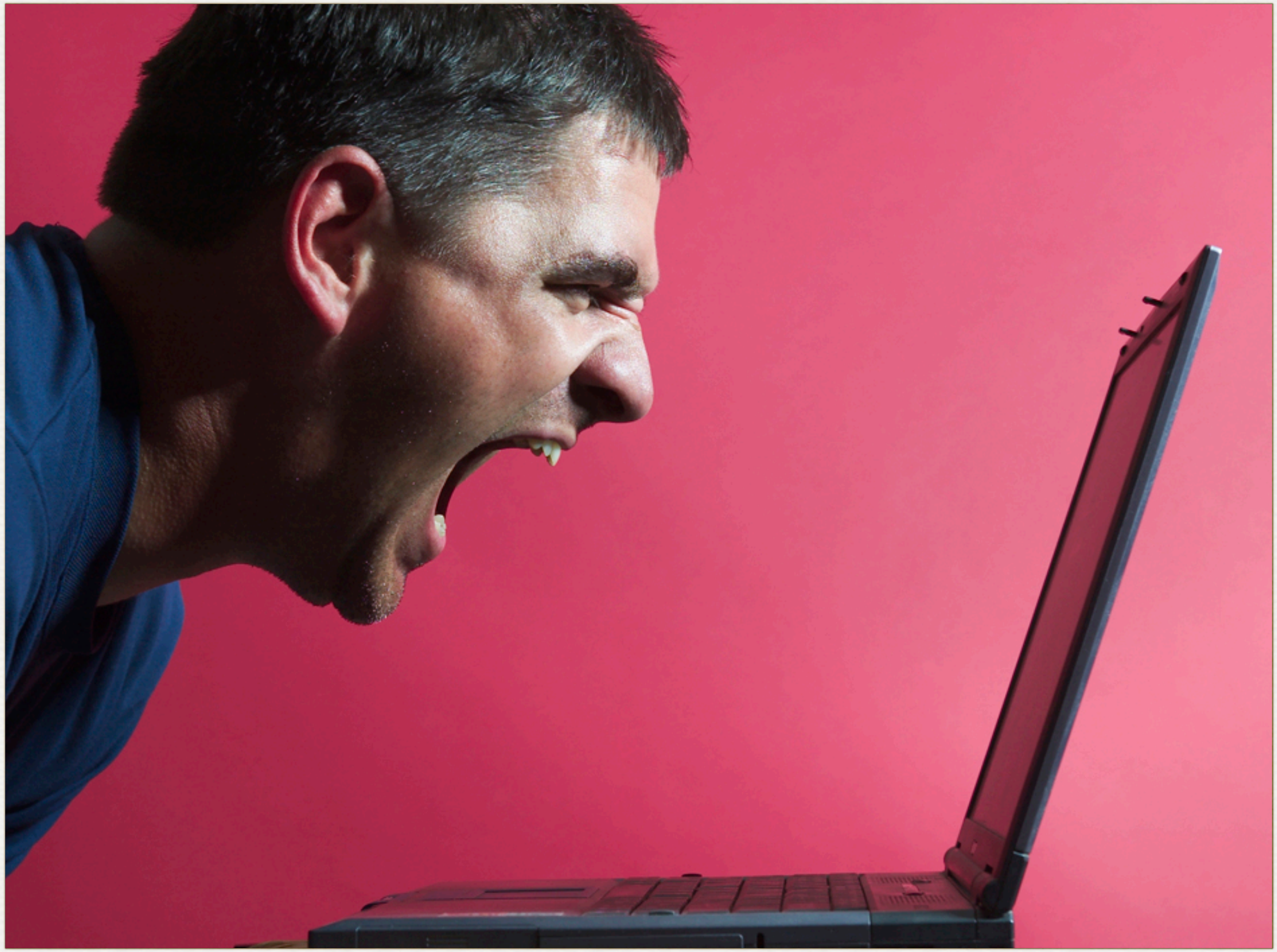
First, Who Am I?

Michael Dinerstein

- Founder/CEO of soon-to-be-launched iPhone app Boundabout. www.boundaboutwith.us

- Started iPhone development 2 years ago with the following book...


SDK 3
Compatible
A complete course in iPhone
and iPod touch programming
Beginning
iPhone 3 Development
Exploring the iPhone SDK
Dave Mark | Jeff LaMarche
Apress



2 years later...

Let's just dive right in...

What is a RESTful web service?

- REST stands for REpresentational State Transfer
- Requirements for a web service:
  - Contains a base URI (http://www.yoursite.com/)
  - Must transfer an “Internet media type” (JSON, XML, YAML, etc.)
  - Must support HTTP methods (GET/PUT/POST/DELETE)
  - API must be hypertext driven

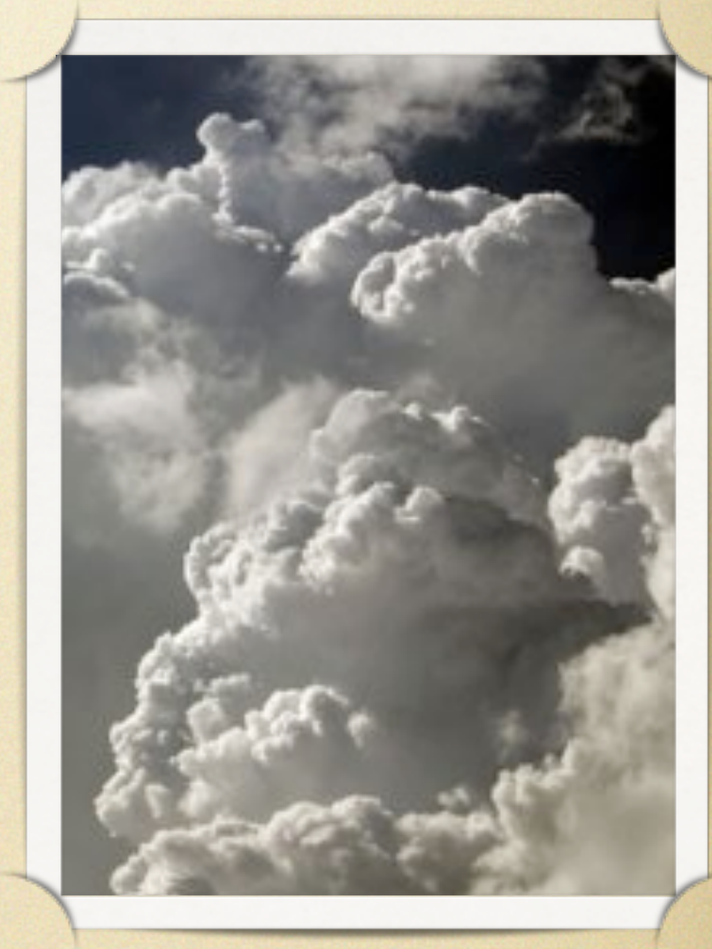
Source: http://en.wikipedia.org/wiki/REST

What you essentially want to do:

iPhone App

http://www.site.com/api/users/3

```
{
    “user_id”: 3,
    “email”: “....”
    “avatar_sm”: “....”
}
```



Web Server

How do we get your app to...

- ...match up your JSON objects with your Core Data model?
- ...know which routes on your web service match up to particular Core Data models?
- ...handle object relationships within Core Data?
- ...prevent double-insertions and maintain the most recent copy of an object?
- ...automatically update views with data recently fetched from your web service?

How do we get your app to...

"JUST WORK"?

Introducing RSAPI

aka REST Simple API

Built on top of the AFNetworking Library

(future versions will be networking library agnostic)

How does it work?

- **RSModelHelper**
  Scaffolding for you to program how your Core Data model relates to your JSON objects.

- **RSAPI**
  Core of the class. Makes API calls, encodes POST / GET parameters, keeps track of API tokens.

- **RSDataFetcher**
  Wrapper for NSFetchedResultsController to automatically update UITableViews with newly fetched data.

User Class

JSON Model from Server	Core Data Model (iPhone)
user_id	userID
name	name
email	email
avatar_sm	avatarSm
friends	friends
group	group

RSModelHelper Overview

- You must directly edit this file to match up to your particular Core Data Model
- Tells RSAPI which JSON properties match up to corresponding Core Data properties.
- Tells RSAPI which JSON property is the object ID.

Matching JSON properties to Core Data properties

```
+(NSDictionary*)jsonPropertyMapForClass:(NSString*)className{
  if ([className isEqualToString:@"YourCoreDataClassName"]){
    return [NSDictionary dictionaryWithObjectsAndKeys:
            @"jsonKey",@"coreDataProperty",
            @"jsonKey_2",@"coreDataProperty_2",
            nil];
  }
  else if ([className isEqualToString:@"User"]){
    return [NSDictionary dictionaryWithObjectsAndKeys:
            @"user_id",@"userID",
            @"avatar_sm",@"avatarSm",
            nil];
  }
  //etc.

  //If we get down here, that means the appropriate model does not
exist.
  [NSException raise:@"Core Data Class Does Not Exist" format:@"Class
Does Not Exist: %@",className];
  return nil;
}
```

User Class

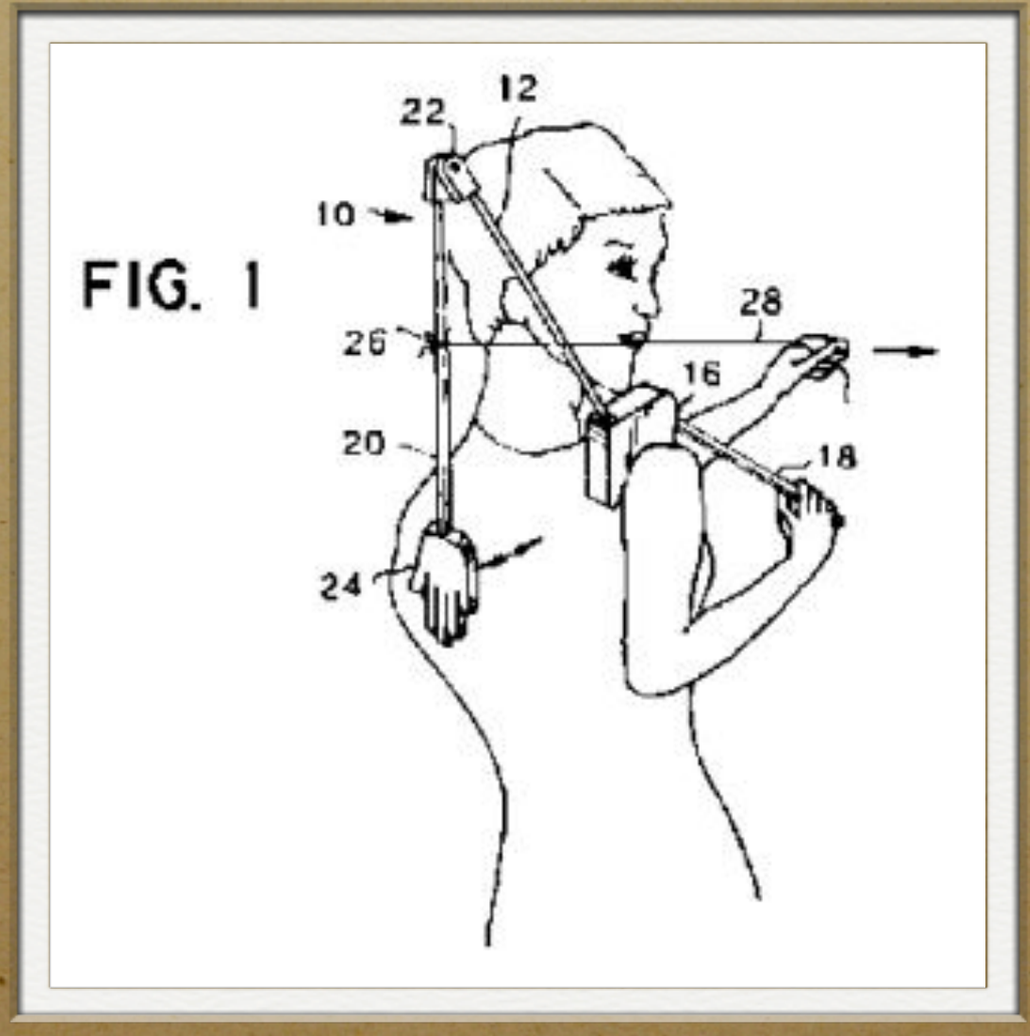
JSON Model from Server	Core Data Model (iPhone)
user_id	userID
name	name
email	email
avatar_sm	avatarSm
friends	friends
group	group

Declare the JSON property that is the Core Data ID

```
+(NSString*)jsonIdKeyForClass:(NSString*)className{
  if ([className isEqualToString:@"User"])
    return @"user_id";
  return @"id";
}
```

Best Practice: ALWAYS try to have the same ID key across all of your JSON models and simply return that value.

Congrats! Your Model is Set Up.



Initializing RSAPI

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
  self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]] autorelease];
  // Override point for customization after application launch.
  self.window.backgroundColor = [UIColor whiteColor];
  [self.window makeKeyAndVisible];

  RSAPI *api = [RSAPI
            setupWithManagedObjContext:self.managedObjectContext
            withPersistentStoreCoord:self.persistentStoreCoordinator
            withManagedObjModel:self.managedObjectModel
            withDevelopmentBase:@"http://localhost:3000"
            withProductionBase:@"http://www.yourproduction.com"
        ];
    ...
}
```

Defaults to Production. To use Development:

```
[api setUseProduction:NO];
```

Setting up Routes

Simple GET and POST Requests

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions{
  ...

  //GET Request
  [api setPath:@"/path/to/object" forClass:@"CoreDataObjectClass"
requestType:RSHTTPRequestTypeGet];

  //POST Request
  [api setPath:@"/api/users" forClass:@"User" requestType:RSHTTPRequestTypePost];
}
```

Setting up Routes

Variables in the URL

```
  [api setPath:@"/api/users/:id" forClass:@"User"
requestType:RSHTTPRequestTypeGet];
  //Pass id = 3: URL resolves to /api/users/3


  [api setPath:@"/api/groups/:id/all_named/:name" forClass:@"Group"
requestType:RSHTTPRequestTypeGet];
  //Pass id = 3 and name = “steve”: URL resolves to /api/groups/3/all_named/steve
```

Setting up Routes

Bypass Core Data Synchronization

```
//Skip Core Data Synchronization and return simply the raw JSON
[api setPath:@"/path/to/data" forClass:@"DATA"
requestType:RSHTTPRequestTypeGet];
```

Setting up Routes

Path Returning More Than 1 Class of Object

```
  //A route that returns many object types
  [api setPath:@"/path/to/objects" forClass:@"MANY"
requestType:RSHTTPRequestTypeGet];
```

JSON Structure Example

```
[{
  "User": [{
    "id": 3,
    "email": "email@email.com",
  }],
  “AnotherClassName”: [{
    “id”: 4,
    “obj_key”: “obj_val”
  },{
    “id”: 5,
    “obj_key”: “obj_val”
  }]
}]
```

Now You Can Make Data Calls!

Basic Call

```
RSAPI *api = [RSAPI sharedAPI];
[api call:@"/path/to/object" params:nil withDelegate:self];
```

RSAPI calls your server, takes the return object, processes it, and automatically saves it to Core Data.

```
-(void)apiDidReturn:(id)arrOrDict forRoute:(NSString *)action{
  // Handle successful call. arrOrDict is the raw JSON.
  // Great place to create an NSFetchedRequest to get your new objects.
}


-(void)apiDidFail:(NSError *)error forRoute:(NSString*)action{
  // Handle unsuccessful call
}
```

Call With Parameters

```
RSAPI *api = [RSAPI sharedAPI];

//Set up your parameters
NSDictionary *strObjDict = [RSAPI encodeObject:@"stringObject"];
NSDictionary *intObjDict = [RSAPI encodeObject:[NSNumber numberWithInt:3]];
NSDictionary *intObjInURL = [RSAPI encodeObjectAsURLParam:[NSNumber
numberWithInt:1]];
NSDictionary *dataDict = [RSAPI encodePostedData:[NSData data]
withFilename:@"your_filename.ext"];      //ONLY for POST requests

//Insert parameters in a dictionary that matches up with server-side variable
names
NSDictionary *sendToServerDict = [NSDictionary
dictionaryWithObjectsAndKeys:strObjDict,@"name",intObjDict,@"number",intObjIn
URL,@"id",dataDict,@"picture",nil];

//Call the API
[api call:@"/api/users/:id" params:sendToServerDict withDelegate:self];
```

A Note About Relationships

- There are two ways to set up a many-to-one relationships

Include child attributes	Pass just the relationship ID

[{
“user_id”: 3,
“group”: {
“id”: 1,
“formed”: 2012-03-16
}
}]
	[{
	“user_id”: 3,
	“group_id”: 1
	}]

- Both will link the user(id=3) to group(id=1)

A Note About Relationships

- Same thing for a one-to-many relationship

Include child attributes	Pass just the relationship ID

[{
“user_id”: 3,
“friends”: {
“id”: 1,
“name”: “name”
},{
“id”: 2,
“name”: “name”
}]
}]
	[{
	“user_id”: 3,
	“friends”: [{
	“id”: 1
	},{
	“id”: 2
	}]
	}]

Advanced Calls

Using an API Token

```
RSAPI *api = [RSAPI sharedAPI];
[api setPath:@"/api/users/login" forClass:@"User"  //or @”DATA”
requestType:RSHTTPRequestTypeGet];


RSAPI *api = [RSAPI sharedAPI];

NSDictionary *email = [RSAPI encodeObject:@"name@name.com"];
NSDictionary *password = [RSAPI encodeObject:@"password"];

NSDictionary *sendToServerDict = [NSDictionary
dictionaryWithObjectsAndKeys:email,@"email",password,@"password",nil];

[api call:@"/api/users/login" params:sendToServerDict withDelegate:self];



-(void)apiDidReturn:(id)arrOrDict forRoute:(NSString *)action{
  NSDictionary*retDict = (NSDictionary*)arrOrDict;
  NSString *token = [retDict objectForKey:@"api_token"];
  RSAPI *api = [RSAPI sharedAPI];
   [api setAPIToken:token named:@"token_param_name"];
}
```

Using an API Token

- RSAPI will automatically:
  - Append ?token_param_name=<token> to GET requests
  - Add token_param_name=<token> to all POST requests
  - Remember the API token the next time the user starts up the app by storing it in NSUserDefaults

Storing Other Tokens

- Say you need to leverage Facebook's functionality and you want to save a user's Facebook Auth Token. Use RSAPI because:
  - Centralized access to all tokens
  - -(void)logout method will remove all Core Data and tokens you set, making it secure

Storing Other Tokens

```objc
//Set a token
[api setToken:@"<FacebookToken>" forKey:@"facebookAuthKey"];
[api setToken:@"<FacebookExpirationDate>" forKey:@"facebookExpDate"];
```

```objc
//Get a token
NSString *fbToken =[api tokenForKey:@"facebookAuthKey"];
NSString *fbExp =[api tokenForKey:@"facebookExpDate"];
```

Handling Views with Multiple Requests

- Say you have a class that makes two calls:
  - One call returns an object
    - The other call returns DATA that needs to be specifically processed.
- You must differentiate them in the -apiDidReturn:forRoute: method.

Handling Views with Multiple Requests

```
-(void)apiDidReturn:(id)arrOrDict forRoute:(NSString *)action{
  if ([action matches:@"/path/to/data"]){
    //Do your processing for the data
  }
}
```

- action returns the path with the URL variables substituted out.
- Added NSString method that matches your route to the returned action route

RSDataFetcher

Automatically Updating UITableView

- NSFetchedResultsController is an imperfect solution for automatically updating UITableViews
- RSDataFetcher is more tightly integrated with RSAPI to provide enough flexibility and out-of-the-box functionality.

Using RSDataFetcher

```
//Get your context and set up a fetchRequest
NSManagedObjectContext *context = [(AppDelegate*)[[UIApplication
sharedApplication] delegate] managedObjectContext];

NSFetchRequest *fetchRequest = [[NSFetchRequest alloc]
initWithEntityName:@"Tour"];
[fetchRequest setPredicate: [NSPredicate predicateWith...]];
[fetchRequest setSortDescriptors:[NSArray arrayWithObject:[NSSortDescriptor
sortDescriptorWithKey:@"aKey"]]];


RSDataFetcher *tourDataFetcher = [[RSDataFetcher alloc]
initWithFetchRequest:fetchRequest inContext:context withKeyPath:@"aKeyPath"
usingCache:nil inTableView:yourTableView];
[tourDataFetcher setDelegate:self];     //Optional delegate method
[tourDataFetcher performFetch];


//Delegate method
- (void)dataFetcherDidFinishUpdating:(FolioDataFetcher *)dataFetcher{
   //Perform additional actions. Fires after the table has completed updating
}
```

Integrating RSDataFetcher with RSAPI calls

```
//No Integration:
RSAPI *api = [RSAPI sharedAPI];
[api call:@"/path/to/object" params:nil withDelegate:self];
[yourDataFetcher performFetch];

-(void)apiDidReturn:(id)arrOrDict forRoute:(NSString *)action{
    [yourDataFetcher performUpdate];
}



//Tighter Integration:
RSAPI *api = [RSAPI sharedAPI];
[api call:@"/path/to/object" params:nil withDelegate:self withDataFetcher:
yourDataFetcher];
```

Thank You!

RSAPI

GitHub Project Page
https://github.com/loudin/RSAPI

Michael Dinerstein

Contact Me
michael@boundaboutwith.us

Check Out My App
http://www.boundaboutwith.us