

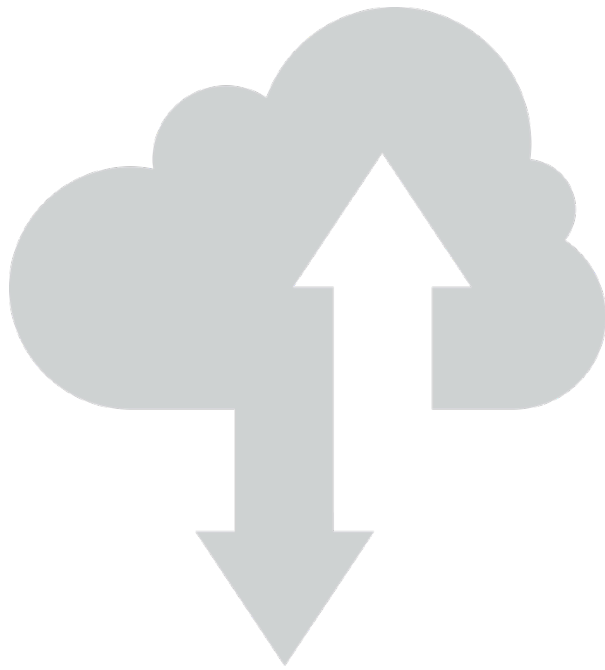
Networking



Brice Wilson

@brice_wilson | www.BriceWilson.net

Overview



Understanding \$http basics

Using \$http shortcut methods

Transformations and Interceptors

Using the \$resource service

\$http Service



Function as a service

Pass it a configuration object

Returns a promise with two additional methods (success and error)

\$http Configuration Object

- method
- url
- params
- data
- headers
- cache



Response Object

- Parameter passed to the “then” functions on the \$http promise
- Contains following properties
 - data
 - status
 - statusText
 - headers
 - config



```
$http({
  method: 'GET',
  url: 'api/books'
})
→ .success(function(data, status, headers, config) {
  // do success stuff
})
→ .error(function(data, status, headers, config) {
  // do error stuff
});
```

Promise Returned from \$http

- Normal promise with the addition of success and error functions
- Each take functions which are passed decomposed HTTP response objects

```
function getAllBooks() {  
  
    return $http({  
        method: 'GET',  
        url: 'api/books'  
    });  
  
}
```

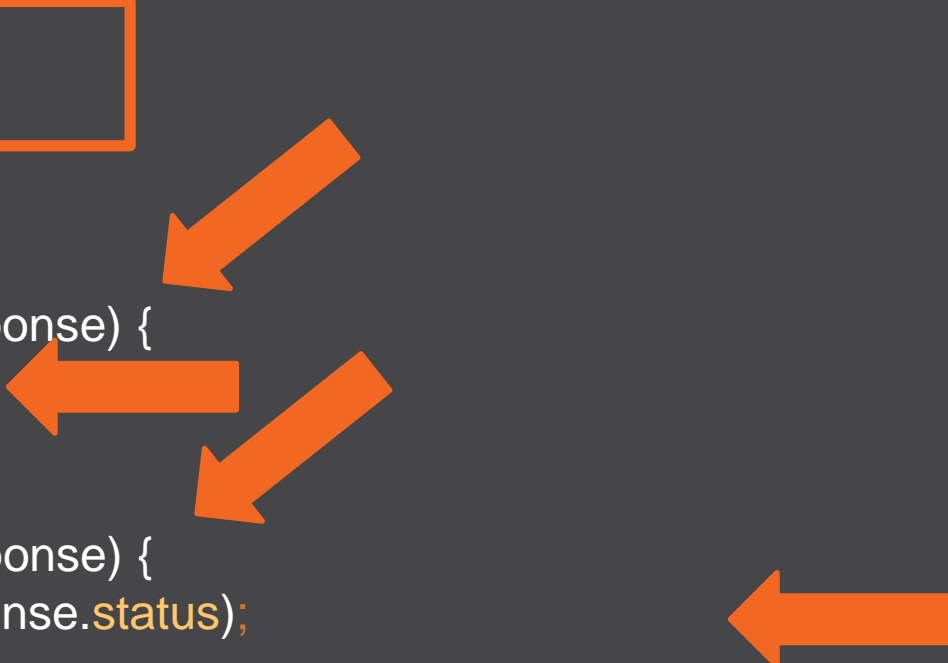
Returning the \$http promise

Forces caller to process http response object

Details about how the data is gathered are leaked to other components

Abstract HTTP Response Away from Caller

```
function getAllBooks() {  
  
    return $http({  
        method: 'GET',  
        url: 'api/books'  
    })  
    .then(sendResponseData)  
    .catch(sendGetBooksError)  
}  
  
function sendResponseData(response) {  
    return response.data;  
}  
  
function sendGetBooksError(response) {  
    return $q.reject('Error: ' + response.status);  
}
```



REST-ful Services

- Representational state transfer
- Web API or HTTP API
- Uses HTTP verbs to specify CRUD operations
- URL conventions address individual as well as collections of resources
- HTTP response codes indicate success/failure of server action

REST-ful CRUD

- Create
 - POST – <http://localhost/api/books>
 - If successful, returns HTTP 201 Created
- Read
 - GET – <http://localhost/api/books> OR <http://localhost/api/books/5>
 - If successful, returns HTTP 200 Success
- Update
 - PUT – <http://localhost/api/books/5>
 - If successful, returns HTTP 204 No Content
- Delete
 - DELETE – <http://localhost/api/books/5>
 - If successful, returns HTTP 204 No Content

\$http Shortcut Methods

- GET
 - `$http.get(url, [config])`
- DELETE
 - `$http.delete(url, [config])`
- POST
 - `$http.post(url, data, [config])`
- PUT
 - `$http.put(url, data, [config])`

Transforming Requests and Responses



Transforming Requests and Responses

```
{  
  author: "Dr. Seuss",  
  title: "The Lorax",  
  year_published: "1971"  
}
```

```
{  
  AuthorName: "Dr. Seuss",  
  BookTitle: "The Lorax",  
  Year: "1971"  
}
```

Transformation

```
{  
  author: "J. R. R. Tolkien",  
  title: "The Hobbit",  
  year_published: "1937"  
}
```

```
{  
  name: "J. R. R. Tolkien",  
  book_title: "The Hobbit",  
  pub: "1937"  
}
```

Using Transformations

- Can be single function or an array of functions
- Default transformations available
 - `$http.defaults.transformRequest`
 - `$http.defaults.transformResponse`
- Override defaults on `$http` configuration object

Using Interceptors

Conceptually
similar to
transformations

Service factories
registered with
\$httpProvider

Two kinds of
interceptors

Two kinds of
rejection
interceptors

\$resource Service

- Abstraction layer above \$http
- Works with REST-ful services
- Returns resources with commonly used server methods already implemented
- Allows quick and easy connections to compatible services



\$resource Classes and Instances

// Create a resource "class"

```
var Books = $resource('/api/books/:book_id');
```



// Create a resource "instance"

```
var book = Books.get({ book_id: 5 });
```



\$resource Properties and Methods

Classes

- `get()`
- `save()`
- `query()`
- `remove()`
- `delete()`
- Custom methods

Instances

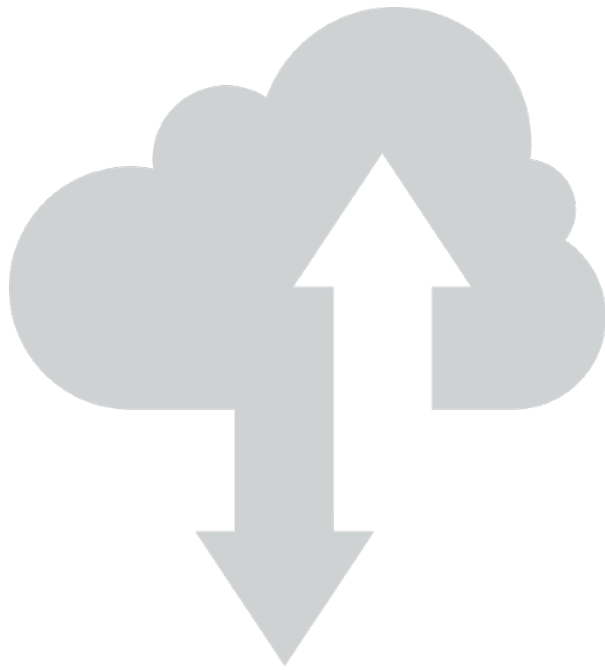
- Entity properties (title, author, etc.)
- `$promise`
- `$resolved`
- `$get()`
- `$save()`
- `$query()`
- `$remove()`
- `$delete()`
- Custom methods (with \$ prefix)

\$resource as Data Service

Quick and easy

Simple CRUD
operations

Summary



How to use the `$http` service

Abstracting web service details away from controllers

Assert control over requests and responses when necessary

Working at a higher level with `$resource`