Building a Killer REST Client for Your REST+JSON API

Les Hazlewood @lhazlewood
Apache Shiro Project Chair
CTO, Stormpath stormpath.com

Stormpath

Stormpath.com

- User Management and Authentication API
- Security for ***your*** applications
- User security workflows
- Security best practices
- Developer tools, SDKs, libraries

Overview

- Resources
- Public / Private API
- Proxy Design
- Active Record
- Fluent API
- Configuration
- Caching
- Authentication
- Pluggability
- Lessons Learned

HATEOAS

- **H**ypermedia
- **A**s
- **T**he
- **E**ngine
- **O**f
- **A**pplication
- **S**tate

Stormpath

Resources

Resources

- Nouns, not verbs
- Coarse-grained, not fine-grained
- Support many use cases
- Globally unique HREF

Collection Resource

- Example:
  `/applications`
- First class resource w/ own properties:
  - `offset`
  - `limit`
  - `items`
  - `first, next, previous, last`
  - etc
- `items` contains instance resources

Instance Resource

- Example:

  `/applications/8sZxUoExA30mP74`

- Child of a collection
- RUD (no Create - done via parent collection)

Translating to Code

Resource

```
public interface Resource {
   String getHref();
}
```


Stormpath

Instance Resource

```
public interface Application
  extends Resource, Saveable, Deleteable {
  ...
}

public interface Saveable {
  void save();
}

public interface Deletable {
  void delete();
}
```

Collection Resource

```
public interface
  CollectionResource<T extends Resource>
    extends Resource, Iterable<T> {

  int getOffset();

  int getLimit();

}
```


Stormpath

Example: ApplicationList

```
public interface ApplicationList
  extends CollectionResource<Application> {
}
```

Design!

Encapsulation

- Public API
- Internal/Private Implementations
- Extensions

- Allows for change w/ minimal impact

  http://semver.org

Encapsulation in practice

```
project-root/
|- api/
|   |- src/main/java
|
|- impl/
|   |- src/main/java
|
|- extendsions/
|   |- src/main/java
|
|- pom.xml
```

Public API

Public API

- *All* interfaces
- Helper classes with static methods
- Builder interfaces for configuration

- NO IMPLEMENTATIONS EXPOSED

Example interfaces

- Client
- ClientBuilder
- Application
- Directory
- Account
- Group
- etc

Classes with static helper methods

```
Client client = Clients.builder()
    ...
    .build();
```

- Create multiple helper classes
  - separation of concerns

Stormpath

Builder interfaces for configuration

```
Client client = Clients.builder().setApiKey(
  ApiKeys.builder().setFileLocation(
    "$HOME/.stormpath/apiKey.properties")
    .build())
  .build();
```

```
Clients.builder() → ClientBuilder
ApiKeys.builder() → ApiKeyBuilder
```

Single Responsibility Principle!

Private API

- Implementations + SPI interfaces
- Builder implementations
- Implementation Plugins

Resource Implementations

- Create a base AbstractResource class:
  - Map manipulation methods
  - Dirty checking
  - Reference to DataStore
  - Lazy Loading
  - Locks for concurrent access

- Create abstract InstanceResource and CollectionResource implementations

- Extend from InstanceResource or CollectionResource

Resource Implementations

```
public class DefaultAccount extends InstanceResource
  implements Account {

  @Override
  public String getName() {
    return (String)getProperty("name");
  }

  @Override
  public Account setName(String name) {
    setProperty("name", name);
    return this;
  }
}
```

Usage Paradigm

Stormpath

Account JSON Resource

```
{
  “href”: “https://api.stormpath.com/v1/accounts/x7y8z9”,
  “givenName”: “Tony”,
  “surname”: “Stark”,
  …,
  “directory”: {
    “href”: “https://api.stormpath.com/v1/directories/
g4h5i6”
  }
}
```

Naïve Design (typesafe language)

```
//get account
String href = “https://api.stormpath.com/v1/....”;
Map<String,Object> account =
  client.getResource(href);

//get account’s parent directory via link:
Map<String,Object> dirLink = account.getDirectory();
String dirHref = (String)dirLink.get(“href”);

Map<String,Object> directory =
  client.getResource(dirHref);
System.out.println(directory.get(“name”));
```

Naïve Design (typesafe language)

- Results in *huge* amount of Boilerplate code
- Not good
- Find another way

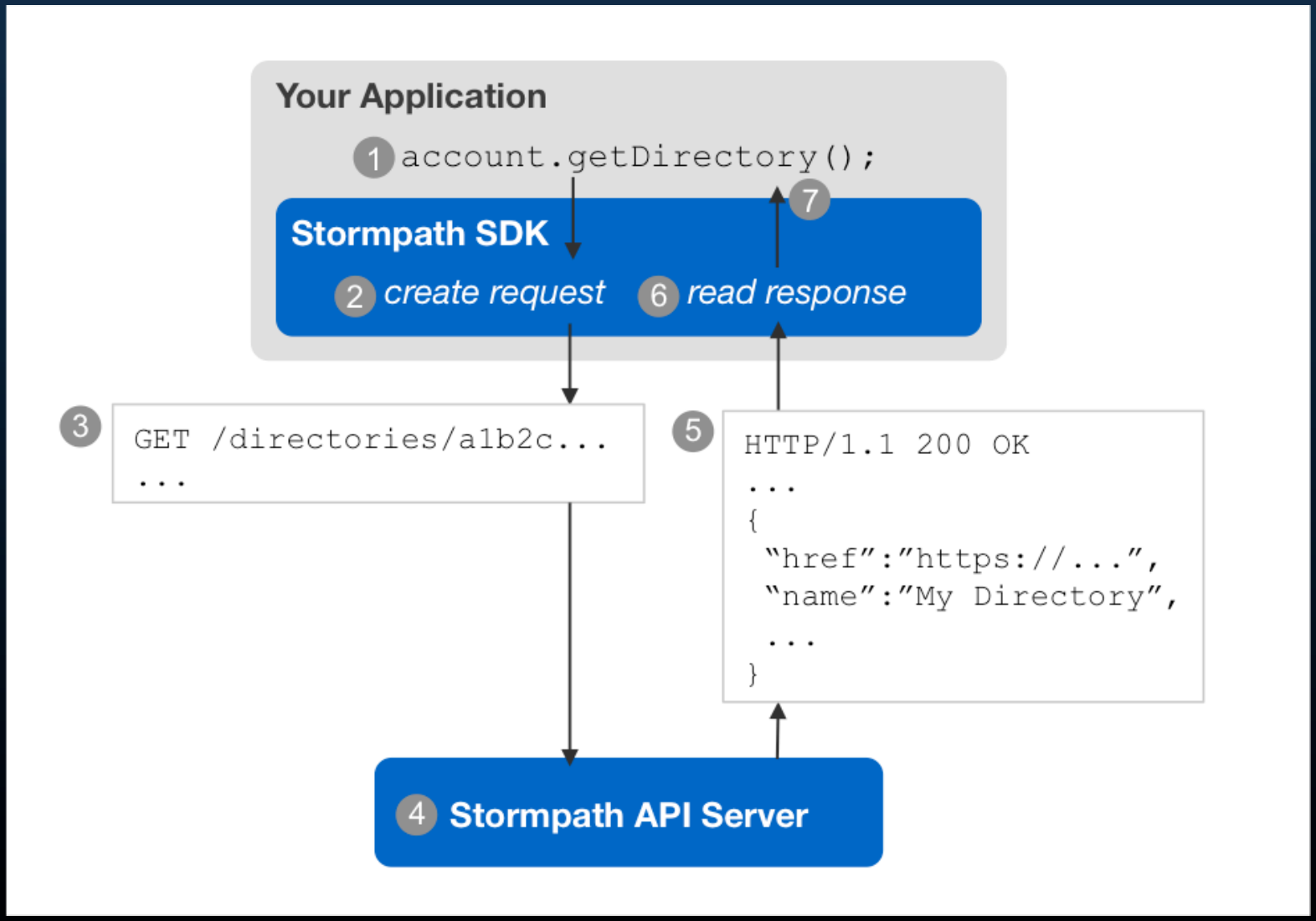
Proxy Pattern

```
String href = “https://api.stormpath.com/v1/....”;
Account account = client.getAccount(href);

Directory directory = account.getDirectory();

System.out.println(directory.getName());
```

Proxy Pattern



Stormpath

Component Design

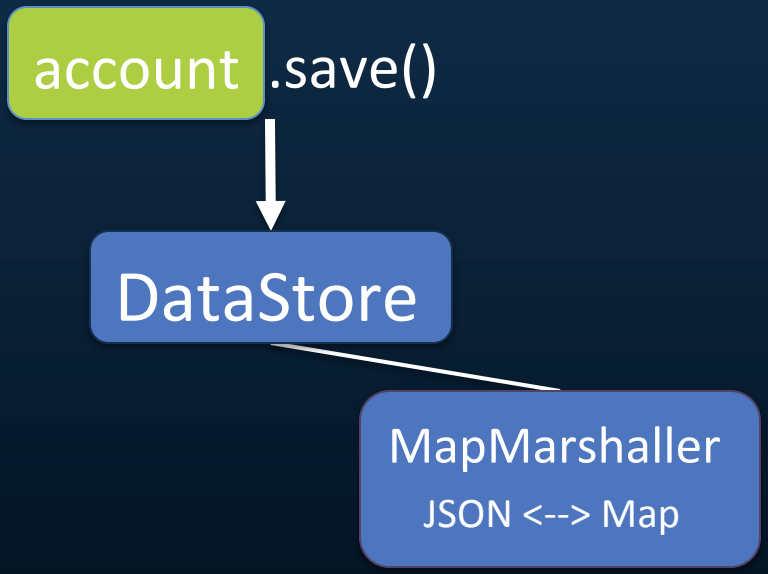
Component Architecture

account.save()

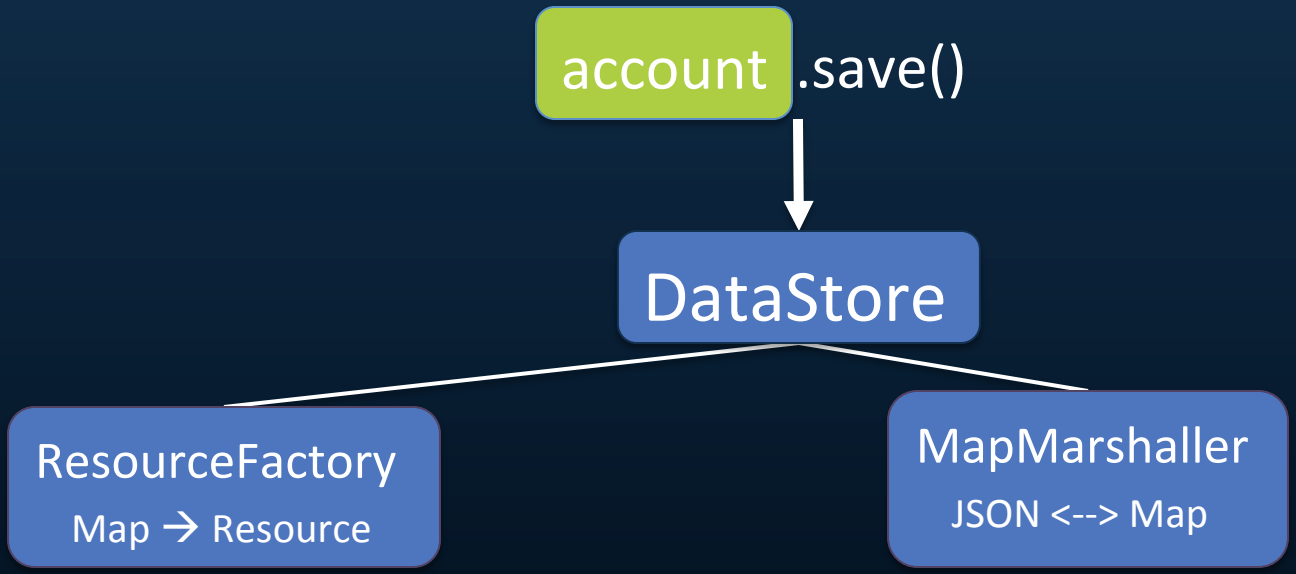
Stormpath

Component Architecture

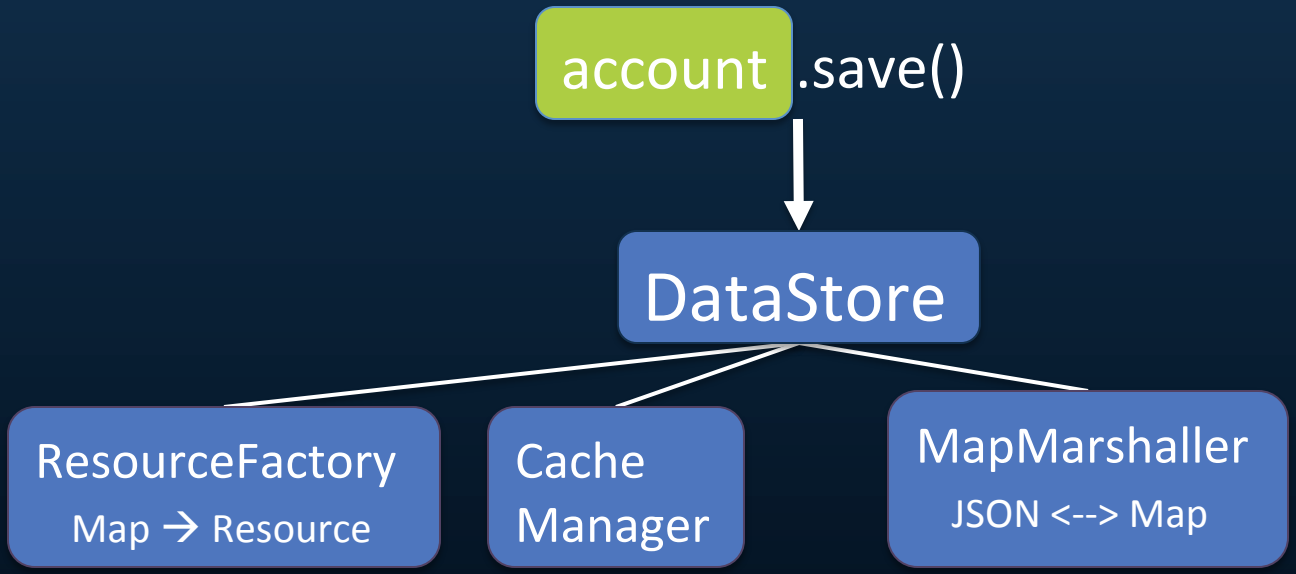
Component Architecture



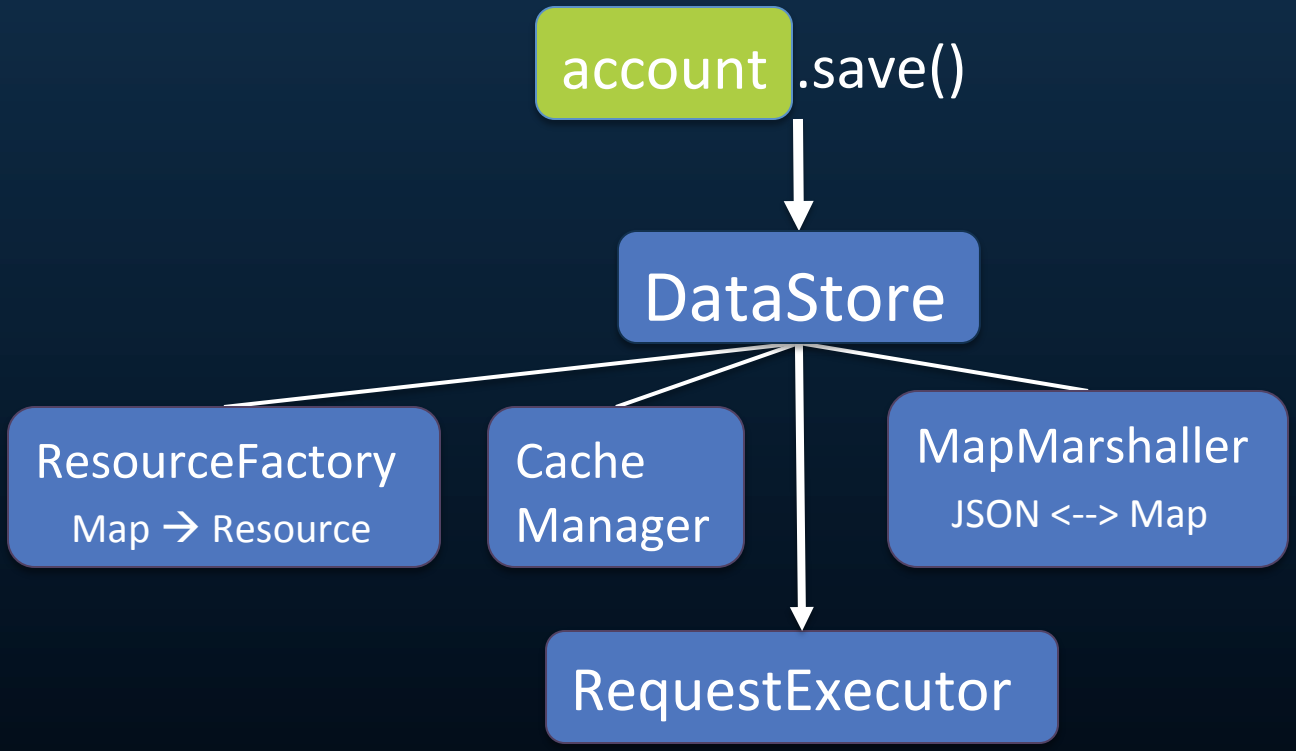
Component Architecture



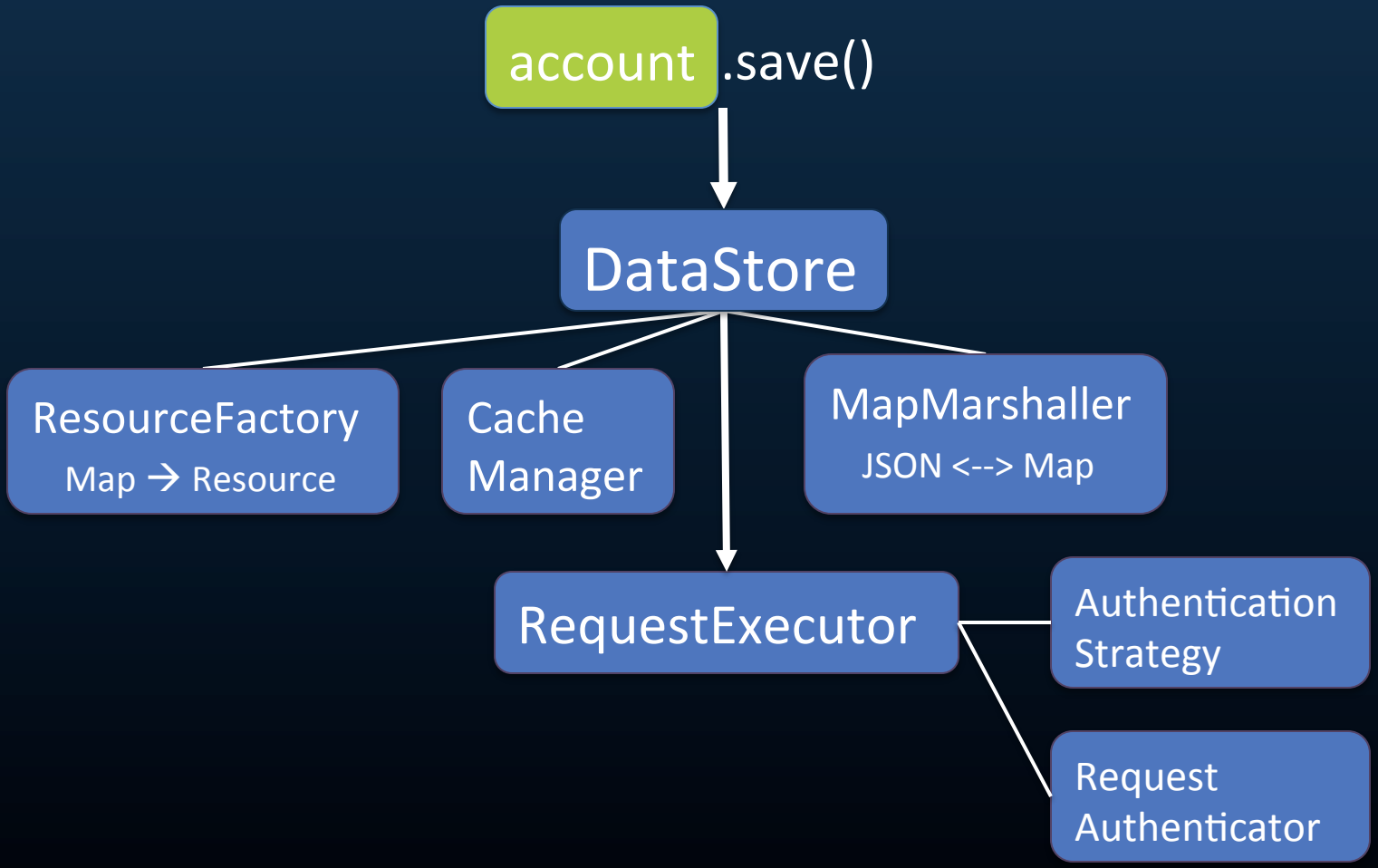
Component Architecture



Component Architecture

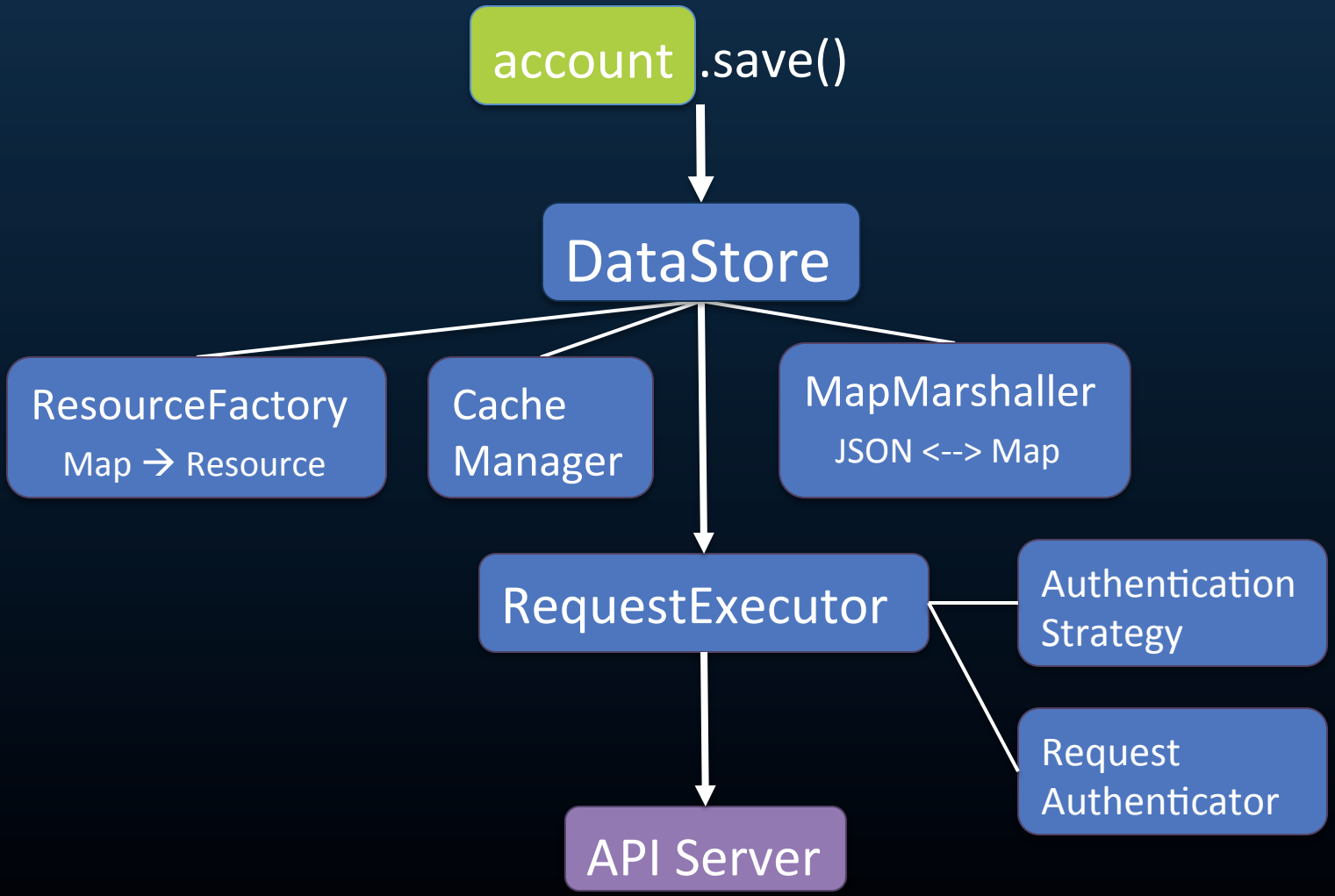


Component Architecture



Stormpath

Component Architecture



Stormpath

Caching

Caching

```
public interface CacheManager {
  Cache getCache(String regionName);
}

public interface Cache {
  long getTtl();
  long getTti();
  ...
  Map<String,Object> get(String href);
  ... other map methods ...
}
```

Caching

```
Account account = client.getAccount(href);

//DataStore:

Cache cache = cacheManager.getCache("accounts");
Map<String,Object> accountProperties = cache.get(href);
if (accountProps != null) {
  return resourceFactory.create(Account.class, props);
}

//otherwise, query the server:
requestExeuctor.get(href) ...
```

Queries

Stormpath

Queries

```
GroupList groups = account.getGroups();
//results in a request to:
//https://api.stormpath.com/v1/accounts/a1b2c3/groups
```

- What about query parameters?
- How do we make this type safe?

Queries

Use a Fluent API!

Queries

```
GroupList groups = account.getGroups(Groups.where()
  .name().startsWith("foo")
  .description().contains("test")
  .orderBy("name").desc()
  .limitTo(100)
);
//results in a request to:

https://api.stormpath.com/v1/accounts/a1b2c3/groups?
  name=foo*&description=*test*&orderBy=name
%20desc&limit=100
```

Queries

Also support simple map for dynamic languages, for example, groovy:

```
def groups = account.getGroups([name: 'foo*',
  description:'*test*', orderBy:'name desc', limit: 100]);

//results in a request to:
https://api.stormpath.com/v1/accounts/a1b2c3/groups?
  name=foo*&description=*test*&orderBy=name
%20desc&limit=100
```

Authentication

Authentication

- Favor a digest algorithm over HTTP Basic
  - Prevents Man-in-the-Middle attacks (SSL won't guarantee this!)

- Also support Basic for environments that require it (Dammit Google!)
  - ONLY use Basic over SSL

- Represent this as an `AuthenticationScheme` to your `ClientBuilder`

Authentication

- `AuthenticationScheme.SAUTHC1`
- `AuthenticationScheme.BASIC`
- `AuthenticationScheme.OAUTH10a`
- `... etc ...`

```
Client client = Clients.builder()
  ...
  //defaults to SAUTHC1
  .setAuthenticationScheme(BASIC)
  .build();
```

Client uses a `Sauthc1RequestAuthenticator` or `BasicRequestAuthenticator` or `OAuth10aRequestAuthenticator`, etc.

Plugins

Stormpath

Plugins

- Plugins or Extensions module
- One sub-module per plugin
- Keep dependencies to a minimum

```
extensions/
|- httpclient
    |- src/main/java
```


Stormpath

Lessons Learned

Lessons Learned

- Recursive caching if you support resource expansion

- Dirty checking logic is not too hard, but it does add complexity. Start off without it.

Lessons Learned: Async, Async!

- Async clients can be used synchronously easily, but not the other way around
- Vert.x, Netty, Scala, Clojure, etc. all require async – hard to use your SDK otherwise
- Netty has a *great* Async HTTP Client that can be the base of your client SDK

Lessons Learned: Async!

```
account.req().groups().where()...
.execute(new ResultListener<GroupList>() {
  onSuccess(GroupList groups){...}
  onFailure(ResourceException ex) {...}
}
```

```
account.req() -> RequestBuilder
execute -> async call w/ promise callback
```

Lessons Learned: Sync

Sync is still easy:

`account.getGroups()` just delegates to:

```
account.req().groups()... .get();
```


Stormpath

Code

```
$ git clone https://github.com/stormpath/
stormpath-sdk-java.git

$ cd stormpath-sdk-java

$ mvn install
```

Thank You!

- les@stormpath.com
- Twitter: @lhazlewood
- http://www.stormpath.com