Principles of REST API Design

Presented By: Amy Wasik
Two Sigma Investments, LP
Software Architecture

Single Server

Services

REST Services
Our Application

Host

Job

Job

Job

Job

Application
Our Application
Our Application

- Client API
- Scheduler
- Host
- Job
- Monitor

Client

Host

May 19, 2017
Software Architecture

- Single Server
- Services
- REST Services
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Services Architecture

- Low Coupling
- Maintainable
- Interoperable
- Language Agnostic
- Shareable
- Scalable
- Resilient
Communication

Single Server: Local API Method Calls

Services: Inter-process Communication (IPC)
Inter-Process Communication (IPC)

- Remote Procedure Calls (RPC)
  - Use a library to convert local calls to remote ones

```java
public interface Jobsystem {
    Job createAJob(JobDetails details);
    void submitJob(Job j);
    List<Job> getJobs(String namePattern);
    List<Job> getMyJobs(String user);
    List<Job> getJobsOther(String query);
    Job getJob(int id);
    void updateJob(JobDetails details);
}
```
Software Architecture

- Single Server
- Services
- REST Services
REST Architecture

- Representational State Transfer (REST)

<table>
<thead>
<tr>
<th>Additional Constraints</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateless</td>
<td>Scalability</td>
</tr>
</tbody>
</table>
REST Architecture

- Representational State Transfer (REST)

<table>
<thead>
<tr>
<th>Additional Constraints</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateless</td>
<td>Scalability</td>
</tr>
<tr>
<td>Cacheable</td>
<td>Increased Capacity</td>
</tr>
</tbody>
</table>
REST Architecture

• Representational State Transfer (REST)

<table>
<thead>
<tr>
<th>Additional Constraints</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateless</td>
<td>Scalability</td>
</tr>
<tr>
<td>Cacheable</td>
<td>Increased Capacity</td>
</tr>
<tr>
<td>Layered</td>
<td>Low Coupling/Interoperability</td>
</tr>
</tbody>
</table>
## REST Architecture

- Representational State Transfer (REST)
  - API Constraints

<table>
<thead>
<tr>
<th>4 Levels of Adherence</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – HTTP Transport</td>
<td></td>
</tr>
<tr>
<td>1 – Resource Oriented Design</td>
<td></td>
</tr>
<tr>
<td>2 – HTTP Verbs as actions on resources</td>
<td></td>
</tr>
<tr>
<td>3 – Hypertext as the Engine of Application State (HATEOAS)</td>
<td></td>
</tr>
</tbody>
</table>

https://martinfowler.com/articles/richardsonMaturityModel.html
REST Architecture

- Representational State Transfer (REST)
  - API Constraints

<table>
<thead>
<tr>
<th>4 Levels of Adherence</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – HTTP Transport</td>
<td>Standard Interface</td>
</tr>
<tr>
<td>1 – Resource Oriented Design</td>
<td>Easier-to-Use API</td>
</tr>
<tr>
<td>2 – HTTP Verbs as actions on resources</td>
<td>Complete API</td>
</tr>
<tr>
<td>3 – Hypertext as the Engine of Application State (HATEOAS)</td>
<td>Easy-to-Learn API</td>
</tr>
</tbody>
</table>

https://martinfowler.com/articles/richardsonMaturityModel.html
REST Level 0

HTTP Transport

- Readable object encoding (typically JSON)
- Standard URI format
public interface Jobsystem {
    Job createAJob(JobDetails details);
    void submitJob(Job j);
    List<Job> getJobs(String namePattern);
    List<Job> getMyJobs(String user);
    List<Job> getJobsOther(String query);
    Job getJob(int id);
    void updateJob(JobDetails details);
}

GET http://example.com/createAJob?name=t&...
GET http://example.com/submitJob?id=123
GET http://example.com/getJobs?name=test
GET http://example.com/getMyJobs?user=me
GET http://example.com/getJobs?query=...
GET http://example.com/getJob?id=123
GET http://example.com/updateJob?id=123&
Level 0: RPC over HTTP

GET http://example.com/createAJob?name=t&user=userA...
HTTP/1.1 200 OK
[other headers]
{ "id": 123
}

GET http://example.com/submitJob?id=123
HTTP/1.1 200 OK
[other headers]
{ "error" : "no permission" }
REST Level 1

Resource Oriented Design

• Divide and conquer
• Easy to understand and navigate API

Standard URI Format

• /{resource}
• /{resource}/{resource-id}
• /{resource}/{resource-id}/{sub-resource}
• /{resource}/{resource-id}/{sub-resource}/{sub-resource-id}
Object Oriented Design

GET http://example.com/createAJob?name= 
GET http://example.com/submitJob?id=12 
GET http://example.com/getJobs?name= 
GET http://example.com/getMyJobs?user= 
GET http://example.com/getJobs?query= 
GET http://example.com/getJob?id=123 
GET http://example.com/updateJob?id=123 

GET http://example.com/jobs/create?name=t&user=me 
GET http://example.com/jobs/get?name=test 
GET http://example.com/jobs/getMy?user=me 
GET http://example.com/jobs/get?query=... 
GET http://example.com/jobs/123 
GET http://example.com/jobs/123/update?name=t2... 
GET http://example.com/jobs/123/instances/start 
GET http://example.com/jobs/123/instances
HTTP Verbs Represent Actions

• More complete and structured APIs

Common Verbs

• GET – Read (Nullipotent)
• PUT – Update (Idempotent)
• POST – Create
• DELETE – Remove (Idempotent)
Standard HTTP Response Codes

- Standard results of actions

<table>
<thead>
<tr>
<th>Success</th>
<th>Client Error</th>
<th>Server Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>400</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>401</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
<td>403</td>
</tr>
<tr>
<td></td>
<td></td>
<td>404</td>
</tr>
</tbody>
</table>
HTTP Verbs for Actions

GET http://example.com/jobs/create?name=t&user=me
GET http://example.com/jobs/get?name=test
GET http://example.com/jobs/getMy?user=me
GET http://example.com/jobs/get?query=...
GET http://example.com/jobs/123
GET http://example.com/jobs/123/update?name=t2...
GET http://example.com/jobs/123/instances/start
GET http://example.com/jobs/123/instances
POST http://example.com/jobs -d '{"name":"test", "user": "me", ...}'
GET http://example.com/jobs?name=test
GET http://example.com/jobs?user=me
GET http://example.com/jobs?query=...
GET http://example.com/jobs/123
PUT http://example.com/jobs/123 -d '{"name":"job" ...}'
POST http://example.com/jobs/123/instances
GET http://example.com/jobs/123/instances
HTTP Verbs for Actions

POST http://example.com/jobs
  -d '{"name":"test", "user": "me", ...}'
HTTP/1.1 201 Created
[other headers]
{ "id": 123 }

POST http://example.com/jobs/123/instances
HTTP/1.1 403 Forbidden
[other headers]
{ "errorCode": 10,
  "moreInfo": "no permission to run this job" }
REST Level 3

REST API Documentation and API Discoverability

- Hypertext As The Engine Of Application State (HATEOAS)
  - Adds links to response that indicate useful actions
- Open API
  - Provides language-agnostic way to describe REST API
  - Lots of tooling for automation
# Open API

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/jobs</td>
<td>Add a new job definition to be run on a periodic basis</td>
</tr>
<tr>
<td>GET</td>
<td>/jobs</td>
<td>Get Jobs by query</td>
</tr>
<tr>
<td>GET</td>
<td>/jobs/{jobId}</td>
<td>Find job by jobId</td>
</tr>
<tr>
<td>PUT</td>
<td>/jobs/{jobId}</td>
<td>Update an existing job definition</td>
</tr>
<tr>
<td>DELETE</td>
<td>/jobs/{jobId}</td>
<td>Deletes a job definition</td>
</tr>
<tr>
<td>POST</td>
<td>/jobs/{jobId}/instances</td>
<td>Start running an instance of the job specified by jobId</td>
</tr>
<tr>
<td>GET</td>
<td>/jobs/{jobId}/instances</td>
<td>Get a list of recent instances for a job defined by jobId</td>
</tr>
<tr>
<td>PUT</td>
<td>/jobs/{jobId}/instances/{instanceId}</td>
<td>Change the priority of a running job instance process specified by instanceId</td>
</tr>
<tr>
<td>DELETE</td>
<td>/jobs/{jobId}/instances/{instanceId}</td>
<td>Stop a running instance of a job specified by instanceId</td>
</tr>
</tbody>
</table>
Open API

POST /jobs Add a new job definition to be run on a periodic basis

Parameters

Name | Description
--- | ---
**body** | Job that needs to be scheduled.

Example Value | Model
--- | ---

```json
job {
  name: string *
  tags: string[]
  state: string
  user: string
}
```

Responses

Response content type: **application/json**

Code | Description
--- | ---
403 | Forbidden to create a job as this user
public interface Jobsystem {
    Job createAJob(JobDetails details);
    void submitJob(Job j);
    List<Job> getJobs(String namePattern);
    List<Job> getMyJobs(String user);
    List<Job> getJobsOther(String query);
    Job getJob(int id);
    void updateJob(JobDetails details);
}
Evolution of the API

**jobs** Job definitions

- **POST** `/jobs` Add a new job definition to be run on a periodic basis
- **GET** `/jobs` Get Jobs by query
- **GET** `/jobs/{jobId}` Find job by jobId
- **PUT** `/jobs/{jobId}` Update an existing job definition
- **DELETE** `/jobs/{jobId}` Deletes a job definition
- **POST** `/jobs/{jobId}/instances` Start running an instance of the job specified by jobId
- **GET** `/jobs/{jobId}/instances` Get a list of recent instances for a job defined by jobId
- **PUT** `/jobs/{jobId}/instances/{instanceId}` Change the priority of a running job instance process specified by instanceId
- **DELETE** `/jobs/{jobId}/instances/{instanceId}` Stop a running instance of a job specified by instanceId
Conclusion

- Modern day best practices
  - Services architectures
  - REST APIs
  - Resource Oriented Design
  - Self-documenting code

- Next steps
  - Evolving APIs
  - Complex operations
  - Error handling, Standard response types
Additional Resources

http://swagger.io/
https://martinfowler.com/articles/richardsonMaturityModel.html

Published API Guides:
https://cloud.google.com/apis/design/

Thank you!