

Testing Rest

About Me

- Maruthi R Janardhan
 - Been with IBM, ANZ, HCL-HP,
 - I run my own startup Leviossa - we are into development services and training.
 - Total 16 years programming C, C++, Python, Java, Javascript, Ruby, Perl, PHP, Grails, etc

Web As A Resource

- Roy Fielding - co-founder - apache http server project.
- His doctoral thesis describes REST as an architectural style
- Why is the Web so prevalent and ubiquitous?
- What makes the Web scale?
- How can I apply the architecture of the Web to my own applications?

What's ReST

- It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.
- REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.).
- REST is not a "standard". There will never be a W3C recommendation for REST,

Examples

- Google search API
- Google Maps API
- Facebook People Search API

Addressable Resources

- Every object and resource in your system is reachable through a unique identifier.
- Does not mean we expose the underlying data model as is.
 - Many times we create API models that are a combination of data models
- TestAlert: APIs looking like an exact copy of database has mostly something wrong with it

How is biz logic exposed

- RPC Style interface operation example:
 - `val = calcSum(a,b)`
- REST style interface treating the sum as a resource
 - <http://host/sum?op1=a&op2=b>
 - Cacheable resource
 - Idempotent
- TestAlert: APIs that are using verbs for urls is not following REST guidelines well enough

How is biz logic exposed

OPERATION	RPC (OPERATION)	REST (RESOURCE)
Signup	POST /signup	POST /persons
Resign	POST /resign	DELETE /persons/1234
Read a person	GET /readPerson?personid=1234	GET /persons/1234
Read a person's items list	GET /readUsersItemsList?userid=1234	GET /persons/1234/items
Add an item to a person's list	POST /addItemToUsersItemsList	POST /persons /1234/items
Update an item	POST /modifyItem	PUT /items/456
Delete an item	POST /removeItem?itemId=456	DELETE /items/456

Sometimes They Dont Fit Well

- Sometimes the concept of resource does not fit well.
- Example: google map direction finding rest api could have been called “findDirection” to indicate that its a method
 - However google makes it a resource and calls it “direction”.
A noun is used instead of verb
 - <http://maps.googleapis.com/maps/api/directions/json?origin=Jakkur&destination=Hebbal>
 - This makes cacheing easy with http semantics
- Think of resources as classes (nouns) and not methods (verbs)

Verbs and Meanings

VERB	MEANING	IDEMPOTENT	SAFE	CACHEABLE
GET	Reads a resource	Yes	Yes	Yes
POST	Creates a resource or triggers a data-handling process	No	No	Only cacheable if response contains explicit freshness information
PUT	Fully updates (replaces) an existing resource or create a resource	Yes	No	No
PATCH	Partially updates a resources	No	No	Only cacheable if response contains explicit freshness information
DELETE	Deletes a resource	Yes	No	No

Constrained Interface

- APIs don't have an “action” parameter. The set of operations are pre-defined - GET/POST/PUT etc
- Builds familiarity and semantics
- Improves Interoperability
- Policy based scaling because of predictable behaviour
- TestAlert: Verify idempotency

Characteristics

- Resources, which are identified by logical URLs. Both state and functionality are represented using resources.
- REST interfaces should return user interface content that is independent of how the user interface will be displayed.
- A web of resources, meaning that a single resource should not be overwhelmingly large and contain too fine-grained details. Whenever relevant, a resource should contain links to additional information
- There is no connection state
- Resources should be cacheable whenever possible

Guiding Principles Of REST

- Do not use "physical" URLs: "http://www.acme.com/inventory/product003.xml". A logical URL does not imply a physical file: "http://www.acme.com/inventory/product/003".
- Queries should not return an overload of data. If needed, provide a paging mechanism.
- Even though the REST response can be anything, make sure it's well documented
- Rather than letting clients construct URLs for additional actions, include the actual URLs with REST responses
- GET/PUT/DELETE access requests should be idempotent

Richardson Maturity Model

- Level 0 - SOAP et al - Single verb and single URI
- Level 1 - Resources, but all requests are get/post
- Level 2 - HTTP Verbs and Status Codes
- Level 3 - Hypermedia
- TestAlert: Push for a higher maturity model

HATEOAS

- Hypermedia as the Engine of Application State
- The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers.
- Links in a document also identify state change
- Older applications usually have a list of precanned services they know exist, and they interact with a central directory server to locate these services on the network

Example of HATEOAS

```
{
  "content": [ {
    "price": 499.00,
    "description": "Apple tablet device",
    "name": "iPad",
    "links": [ {
      "rel": "self",
      "href": "http://localhost:8080/product/1"
    } ],
    "attributes": {
      "connector": "socket"
    }
  }, {
    "price": 49.00,
    "description": "Dock for iPhone/iPad",
    "name": "Dock",
    "links": [ {
      "rel": "self",
      "href": "http://localhost:8080/product/3"
    } ],
    "attributes": {
      "connector": "plug"
    }
  } ],
  "links": [ {
    "rel": "product.search",
    "href": "http://localhost:8080/product/search"
  } ]
}
```


Link Data

- rel - indicates the type of link: next, previous, edit
- href - the actual location of the resource
- type - the content-type produced by the url

Pagination With HATEOAS

Clients dont have to construct a set of parameters or URLs for a page.. they just use the urls in current data to reach next set of data

```
<products>
  <link rel="next" href="http://example.com/store/products?
startIndex=5"/>
  <product id="123">
    <name>headphones</name>
    <price>$16.99</price>
  </product>
  ...
</products>
```

REST and Scaling

- Many times modern backends are not consistent
 - CAP Theorem shows that consistent systems do not scale easily
 - We fall back on eventual consistency instead of instantaneous consistency
- Transactions are not perfect in large scale systems
- TestAlert: Test the impact of inconsistent systems

Making A Purchase

- Something like amazon.com need replicated databases for failover (No concept of backups)
- An order placed on one database instance has to be replicated to all.
 - There is a moment when the data is inconsistent on other db
- Idempotency of REST operations like PUT helps just repeat operation on inconsistent instances

Test Plan

- A Test Plan is not just a formality document. It is well thought out written down form of a plan that has to cover most of these aspects
 - Objectives
 - What do you want to achieve with this testing? Functional correctness, performance verification, accessibility conformance?
 - Scope
 - What are you testing? Entire system? One service, just a component or just the database?

Test Plan

- Test Strategy
 - Who will test, how much of it will be manual, what kind of automation, what all will be covered in automation, what is the strategy for any mocking or harnesses to be created, if there is stress testing how we will get the data for it, if we are taking production data, how do we mask things for privacy, etc

Test Plan

- Environments
 - What kind of systems are needed for the test, how it is separated from development boxes, what kind of access is needed to whom, what configuration and scripts are to be run on the boxes, how the build gets there, etc
- Test Schedule
 - When is the test done, how does it sync with the development schedule, how will the bug fix cycle be timed, etc

Test Plan

- Control Procedures
 - Who reviews test cases and when
 - When, how and who all will be in bug review meetings
 - How a change request is going to be raised
 - How the defects are reported and tracked
 - What tracking and reporting tools are going to be used

Test Plan

- Functions
 - This is an elaboration of scope with detail of a list of functionality that will be tested
- People Resources
 - Assign responsibility to various people on team
- Categorisation, Suspension and Exit Criteria
 - How defects are categorised, when the test can be suspended, what are the exit criteria needed for the testing
- Risks
 - Risks of schedule, technical, management, personnel and requirements

Test Cases

- Test Cases highlight the steps to test a particular Scenario.
- Not always is it practical to write these down. Its usually straight away written and documented in code
- Test Suites are a group of related test cases that can be used to form a meaningful group
 - Ex: functional tests for module1, network performance tests.

Web Services Testing Strategies

- Code based testing with things like httpclient/cxf/restassured
- Tools
 - JMeter
 - RestAssured
 - SOAP UI
 - WebInject
 - TestMaker
 - CURL in scripts
 - REST Clients (Postman, Http4e)

CURL

- curl is a tool to transfer data from or to a server, using one of the supported protocols (HTTP included)
- Get requests: curl <http://mysite.com/myresturl>
- PUT request

```
curl -H "Content-Type: application/json" -X PUT --verbose http://mysite.com/myresturl --data-binary "@data.json" -i
```

Example Curl

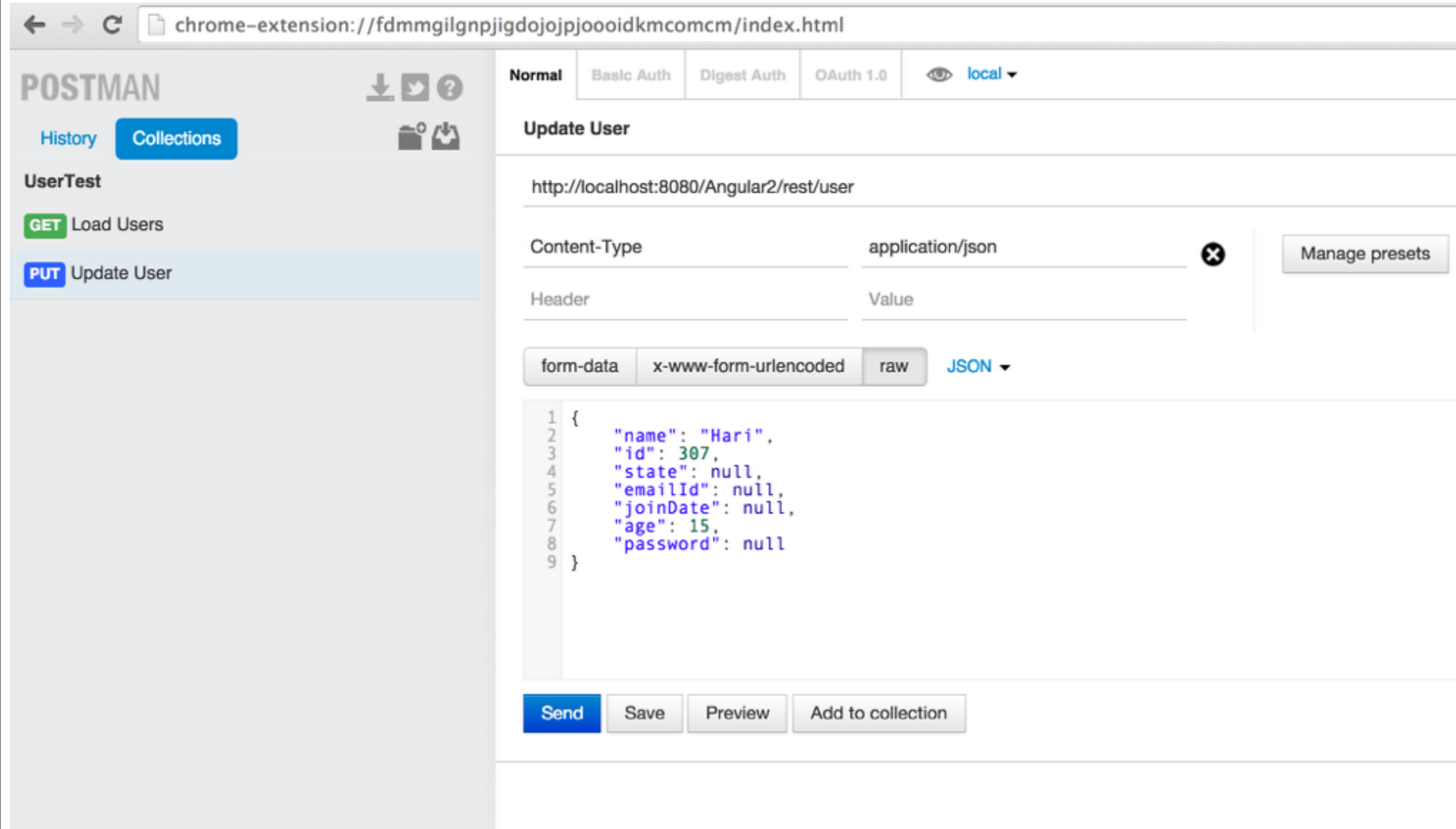
- Obtain the data for a user using curl get on <http://localhost:8080/Angular2/rest/user/{id}>
- Save the response to file using indirection curl url > file
- Edit the file and then do a PUT request to update the data in an user using a syntax like:

```
curl -H "Content-Type: application/json" -X PUT --verbose http://mysite.com/myresturl --data-binary "@data.json" -i
```

postman

- Its a chrome plugin to do REST testing
- Create requests, replay and organise
- Test different environments
- Test responses with javascript with jetpacks (paid addon)
- run postman tests on commandline using newman

Collections



The screenshot displays the Postman web interface within a browser window. The address bar shows the URL: `chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm/index.html`.

Left Sidebar:

- POSTMAN** logo and navigation icons (download, share, help, folders, sync).
- Buttons for **History** and **Collections** (highlighted in blue).
- UserTest** collection name.
- API methods and names: **GET** Load Users and **PUT** Update User (highlighted in light blue).

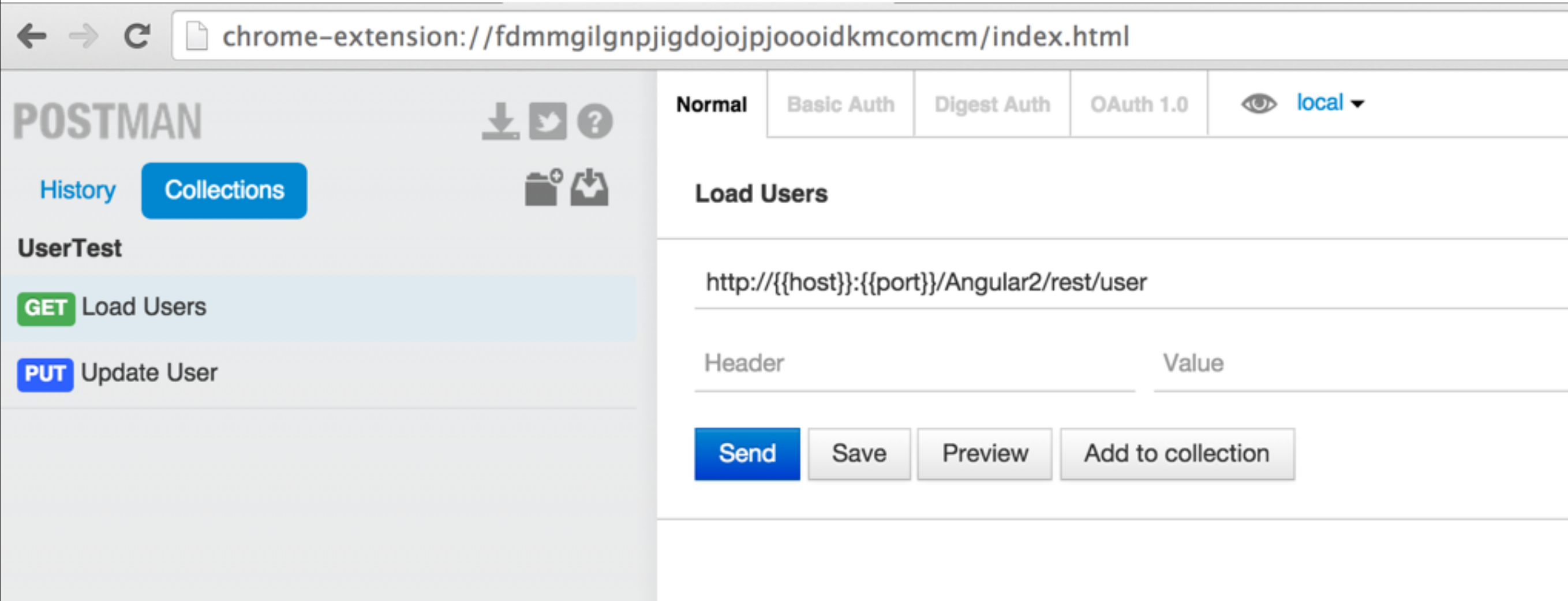
Main Interface:

- Auth:** Normal, Basic Auth, Digest Auth, OAuth 1.0, and a **local** dropdown menu.
- Request Name:** Update User
- URL:** `http://localhost:8080/Angular2/rest/user`
- Content-Type:** `application/json` (with a close icon).
- Header Table:**

Header	Value
--------	-------
- Body Type:** form-data, x-www-form-urlencoded, raw, and **JSON** (selected).
- JSON Body:**

```
1 {
2   "name": "Hari",
3   "id": 307,
4   "state": null,
5   "emailId": null,
6   "joinDate": null,
7   "age": 15,
8   "password": null
9 }
```
- Buttons:** Send, Save, Preview, and Add to collection.
- Manage presets** button.

Environments



The screenshot displays the Postman web interface within a browser window. The address bar shows the URL: `chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm/index.html`.

POSTMAN

Navigation tabs: History, Collections (selected).

UserTest

- GET** Load Users (selected)
- PUT** Update User

Authentication: Normal (selected), Basic Auth, Digest Auth, OAuth 1.0. Environment: local (selected).

Load Users

URL: `http://{{host}}:{{port}}/Angular2/rest/user`

Header	Value
--------	-------

Buttons: Send, Save, Preview, Add to collection.

Postman testing

- Test User retrieval and update using postman
- Create a collection of tests and export/import collections
- Install jetpack and run test collections
- Add test verification scripts
- Verify authorization works using tests

Http4e

- Eclipse plugin for REST API testing
 - Tabs for each API call u want to test
 - Auto suggest for headers and other setup values
 - Raw, Pretty, JSON, HEX, Browser response view
 - SSL/HTTPS and Unicode support
 - Auth and proxy support
 - Import/Export
 - Multipart request support

Use Http4e

- Create multiple requests get/update
- Export as http4e and jmeter scripts
- Use basic auth and parameterization
- Multipart request testing

RestAssured

- A flexible java client library for rest testing

```
<dependency>
  <groupId>com.jayway.restassured</groupId>
  <artifactId>rest-assured</artifactId>
  <version>2.4.1</version>
</dependency>
<dependency>
  <groupId>com.jayway.restassured</groupId>
  <artifactId>json-schema-validator</artifactId>
  <version>2.4.1</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

Basic Structure

- `get("/lotto").then().body("lotto.lottold", equalTo(5));`
- `get("/lotto").then().body("lotto.winners.winnerId", hasItems(23, 54));`
- `when().get("/json").then().body("$", hasItems(1, 2, 3));`
- given...when...then pattern

```
given().accept(ContentType.JSON).authentication()  
    .basic("admin", "admin123")  
    .when()  
        .get("https://localhost:8443/RestSample/services/users")  
    .then()  
        .body("id", hasItems(307, 605, 619));
```

Lets Try It

- Turn off authentication on server, Perform a get request on /users and verify that there are x number of users
 - Matching approach: "\$.size()", is(14)
- Enable authentication and use auth in the request
`.authentication().basic("admin", "admin123")`

JSON Schema Validation

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product set",
  "type": "array",
  "items": {
    "title": "Product",
    "type": "object",
    "properties": {
      "id": {
        "description": "The unique identifier for a product",
        "type": "number"
      },
      "name": {
        "type": "string"
      },
      "price": {
        "type": "number",
        "minimum": 0,
        "exclusiveMinimum": true
      },
      "dimensions": {
        "type": "object",
        "properties": {
          "length": {"type": "number"},
          "width": {"type": "number"},
          "height": {"type": "number"}
        },
        "required": ["length", "width", "height"]
      },
      "warehouseLocation": {
        "description": "Coordinates of the warehouse with the product",
        "$ref": "http://json-schema.org/geo"
      }
    },
    "required": ["id", "name", "price"]
  }
}
```

Lets Try It

- Use json schema to validate that the response contains an array of users and each user has atleast the id,name and age as mandatory fields

Using JSONObject on responses

- Json library allows dealing with and manipulating objects during testing

```
<dependency>  
  <groupId>org.json</groupId>  
  <artifactId>json</artifactId>  
  <version>20141113</version>  
</dependency>
```

```
get(".....").body().asString();  
JSONObject json = new JSONObject(usersJson);  
json.put("name", "Hari");  
System.out.println(json.toString());
```

- Load an user, update and verify

Use Params

- `given().queryParams()`
- `given().pathParam("param",value)`
 - `/user/{param}`
- `given().formParam()`
- Use path params to retrieve a user using path param

Builder Pattern

- Refer to Builder Pattern code on wikipedia

```
StreetMap map = new StreetMap.Builder(new Point(50, 50), new  
                                         Point(100,100))  
    .landColor(Color.GRAY)  
    .waterColor(Color.BLUE.brighter())  
    .build();
```

Using Builder Pattern

```
RequestSpecBuilder builder = new RequestSpecBuilder();
RequestSpecification specWithParam = builder.addFormParam("name",
"value")
    .build();
ResponseSpecBuilder resBuilder = new ResponseSpecBuilder();
ResponseSpecification resSpec =
resBuilder.expectStatusCode(is(200))
    .expectBody("users", hasKey("name")).build();

given(specWithParam, resSpec).get("/someurl");
```

Filters

- Filters are ways to default functionality to requests dynamically across the board.

```
class MyAuthFilter implements Filter{
    public Response filter(FilterableRequestSpecification requestSpec,
        FilterableResponseSpecification responseSpec, FilterContext ctx)
    {
        requestSpec.authentication().basic("admin", "admin123");
        return ctx.next(requestSpec, responseSpec);
    }
}
```

Multipart Data

- Forms can be built to send data in multipart encoded format. Useful for sending files.
- Data is not encoded but data is separated using a boundary

```
<form action="MultipartHandler" method="post" enctype="multipart/  
form-data">
```

```
Name: <input type="text" name="name"/><br>
```

```
Address: <textarea rows="4" cols="100" name="address"></  
textarea><br>
```

```
Photo ID: <input type="file" name="photo"/><br>
```

```
<input type="submit" value="send"/>
```

```
</form>
```

Multipart Data

```
-----5082650181867693407554623250
Content-Disposition: form-data; name="name" Maruthi
-----5082650181867693407554623250
Content-Disposition: form-data; name="address" 5th Avenue
-----5082650181867693407554623250
Content-Disposition: form-data; name="photo";
filename="SmallImage.jpg" Content-Type: image/jpeg
ÿØÿàJFIFHHÿâXICC_PROFILEHLinomintrRGB XYZ Î 1acspMSFTIEC
sRGBöÖÓ-HP
cprtP3desclwtptöbkptrXYZ6789:CDEFGHIJSTUVWXYZcdefghijstuv
wxyzç£œ¥|§¨©ª³
´μ¶·¸¹ºÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëñòóôõö÷øùúÿÄ
-----5082650181867693407554623250
```

System Performance

- Scalable Systems
- Stateless Systems
 - Chaos Monkey
- Throughput and Response times
- System bottleneck finding
- Use case profiling

JMeter

- Basic web test plan with http sampler
 - Using session managers
- Recording web test plan using a proxy
- Parameterizing test plans
- Making the parameters data driven
- Using logical controllers - once only controllers

Extraction and Manipulation

- Setup the json extractors plugin for jmeter
- JsonPath can be used for json extractors to retrieve values from json: <http://goessner.net/articles/JsonPath/>
- Extract the uris from users list using json extractor:
\$..links[0].uri
- Use DebugSampler to view the extracted variables

Extraction and Manipulation

- Use forEach controller to loop over the extracted URIs and fetch each user
- Use json path extractor to extract the age
- Use regex extractor to extract the full response into a variable
- Use jsr223 post processor to replace age (increment by 5)

```
userJson = vars.get('userJson_g0')  
age = vars.get('userAge')  
age = Number(age)+5  
userJson = userJson.replace(/"age":.*?/,g, '"age":'+age+',')  
vars.put('updatedUserData',userJson)
```

Variables in JSR223

- log - (Logger) - can be used to write to the log file
- Label - the String Label
- Filename - the script file name (if any)
- Parameters - the parameters (as a String)
- args[] - the parameters as a String array (split on whitespace)
- ctx - (JMeterContext) - gives access to the context
- vars - (JMeterVariables) - gives read/write access to variables: `vars.get(key);`
`vars.put(key,val);` `vars.putObject("OBJ1",new Object());` `vars.getObject("OBJ2");`
- props - (JMeterProperties - class `java.util.Properties`) - e.g. `props.get("START.HMS");`
`props.put("PROP1","1234");`
- prev - (SampleResult) - gives access to the previous SampleResult (if any)
- sampler - (Sampler)- gives access to the current sampler
- OUT - `System.out` - e.g. `OUT.println("message")`

Extraction and Manipulation

- Use manipulated data as a body in the put request
- Use response assertion to ensure we have a 204 response code
- Verify the update

More Things JMeter

- Using Threading and API test groupings
- Using Java Samplers for custom behaviour
- Distributed Testing
- Http4e exports into JMeter

Jenkins

- Build Automation Tool
- Create jobs that are a series of actions to run in a sequence
- Keeps a history of job runs, who ran them and results
- Plugin architecture allows for a rich set of customization
 - GIT client, JUnit, Maven, Post build deploy scripts