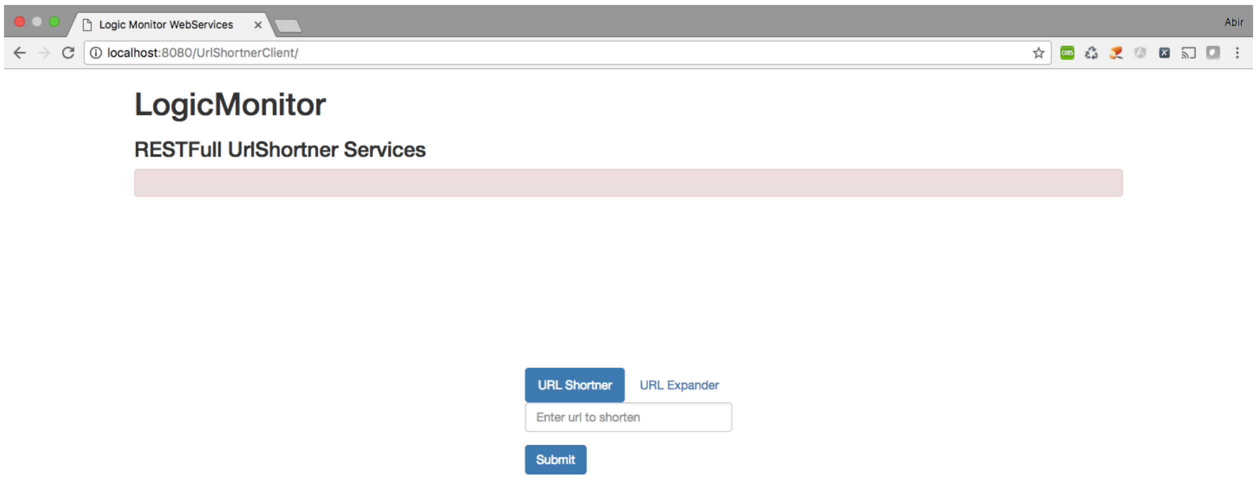
Design Document v1.0
LogicMonitor Url Shortner Service

Abir Lal Saha- Arizona State University

Problem Statement

Create a URlShortner Service which is a web service having two functionalities

1. Given a long url, return a shorten/tiny url.
2. Given the shortened url, return the original long url


Logic Monitor WebServices
localhost:8080/UrlShortnerClient/
Abir
LogicMonitor
RESTFull UrlShortner Services
URL Shortner
URL Expander
Enter url to shorten
Submit

Application Front-End

Application Access Point: localhost:8080/UrlShortnerClient/

[Considering both the war files are deployed correctly in tomcat server]

Requirements

1. Must use Java and simple Java Servlet
2. Create a front-end UI to interact with backend RESTFull web services
3. Shortened url must be unique
4. Shortened url must contain only alphabetical letters and numbers
5. Services must be scalable
6. Capability of blacklist of shortened url

Deliverables

1. UrlShortnerClient.war
2. UrlShortner.war
3. Zipped source code – UrlShortnerClient.zip & UrlShortnerService.zip
4. Design Document
5. Video

Build Instruction

Once you have unzipped the source code and imported the two maven project-UrlShortner Client (REST client to invoke the webservice) and UrlShortnerService (Java webservice) into any IDE or have reached the project directory using terminal, run the following command in both the project directory

```
mvn clean install -DskipTests
```

Deployment Instruction

1. You can deploy the two war file (UrlShortnerClient.war & UrlShortner.war) generated from the maven build or by downloading from the submitted google drive link in to apache tomcat server.

   Deployment Techniques

1. Either integrate apache tomcat into Eclipse or any other IDE and directly deploy your war files in the embedded tomcat.
2. Copy the war files in the webapps folder in apache tomcat home path and restart the server
3. Deploy from Tomcat Manager- You can access the manager dashboard by visiting: http://localhost:8080/manager. Then you can upload the war files by selecting war upload option and restarting the tomcat server.

Design Approach

TinyUrl Generation

Given a long url, a shortened url can be formed using alphabetical letters and numbers[a-zA-z0-9]. So, we have 26+26+10 = 62 different characters to choose from to create a tinyurl.

Suppose, if we decide that the tinyurl would be of length 6. And if we append the random tinyurl generated to a base url like(www.tinyurl.com), we can effectively create 62^6 ~= 56 billion unique urls.

Requirements: The given url should be fully qualified url (contains http:// or https://)

DataStructure

Basically my approach is to store the given url and the generated tinyUrl based on the business logic provided in to a HashMap<String,String> baseMap as a key-value pair. So the key would the longUrl and the value would be the corresponding tinyUrl.

Uniqueness

With each request of tinyUrl creation, the code will first check whether the key(longUrl) already exists in the baseMap or not. If it is present, it will simply return the corresponding tinyUrl. And hence, it prevents from creating tinyUrl for already used longUrls

Again, if a new longUrl used and a 6 character long tinyUrl is generated, we check it against the baseMap whether this value(tinyUrl) is already generated for any other longUrl or not. If it is present, then we regenerate the tinyUrl else we store it into the hashmap.

Data Persistence & Consistency

In order to persist the data and maintain consistency between two service calls, we persist the map data into the file system (map.txt file) and reads it back from the file when required. We can opt for storing the data in the database instead of storing it into in-memory data structures or can store the data in any cloud based storage services.

Scalability

The services have been developed to a scale and can be extended to serve requests in distributed systems. We can persist the data in distributed databases using distributed sharding techniques adding an additional data persistence layer. Multiple machines may be required to store all the mappings and since they will be geographically distributed adopting an incremental id approach would not work.

Cost

Each entry is stored as <LongURL, TINYURL> where TINYURL is a 6-character string. Assuming max URL length is 2000 characters, then each entry takes 6 * 4 bytes + 2000 * 4 bytes = 8.0 KB. If we store a million URL mappings, we need around 8.0GB storage.

Capability of BlackList URLS

We can store all the black list urls in our storage media/ database and we can check whether a generated tinyUrl is blacklisted or not. If it is blacklisted, we can re-generate the tinyUrl.

System Architecture

UrlShortnerCLient

Step 1:

TinyServletForm -----> UrlServletShortenLM
index.jsp doPost()

ExpandServletForm -----> UrlServletExpandLM
index.jsp doPost()

[Creation of UrlLM obj from form-value]

Step 2:

UrlServletShortenLM ----> UrlClient ---- > invokes web services (…/getTinyUrl)
doPost()
UrlServletExpandLM ----> UrlClient ---- > invokes web services (…/getOrgUrl)
doPost()

Step 3:

Webservices--------> UrlCLient --- > UrlServletShortenLM ------ > index.jsp
(send response) shows response on UI

Webservices--------> UrlCLient --- > UrlServletExpandLM ------ > index.jsp
(send response) shows response on UI

UrlShortnerService

Step 1:

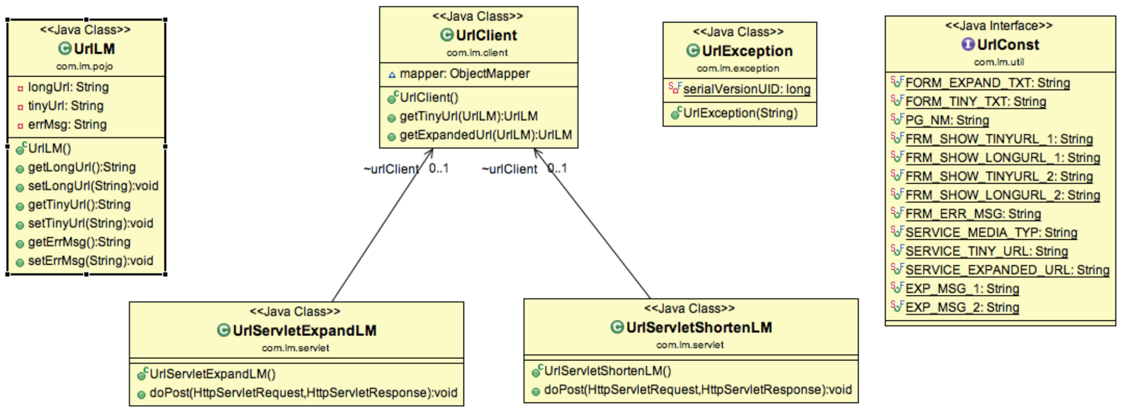
UrlClient------ > UrlServiceLM---→ getTinyUrl()/getExpandedUrl()
Request Invokes corresponding methods based on request url

Step 2:

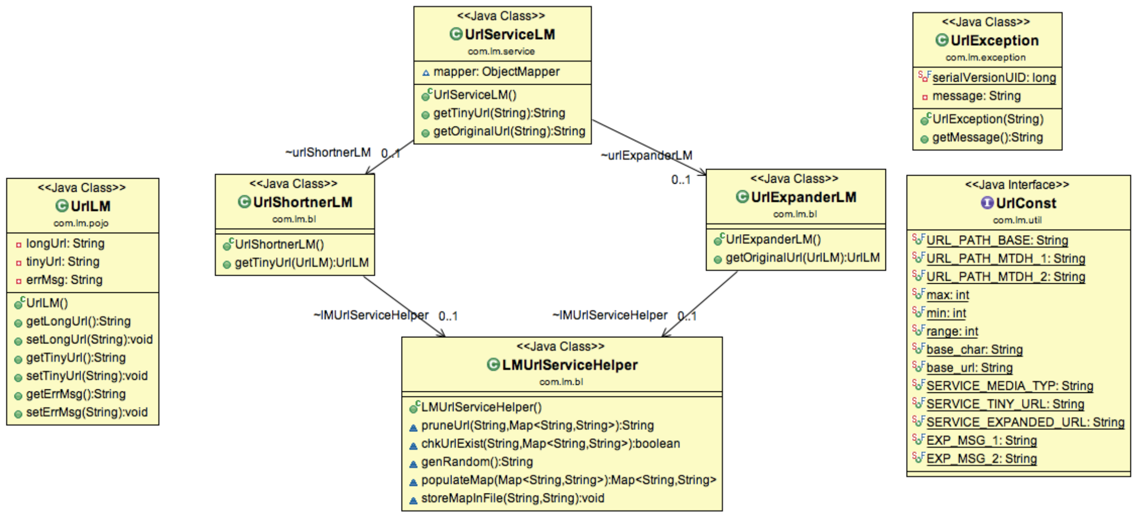
UrlShortnerLM/ UrlExpanderLM ---→ UrlServiceLM ---→ UrlClient
Send response back Send response back to client

Class Diagram

UrlShortnerCLient

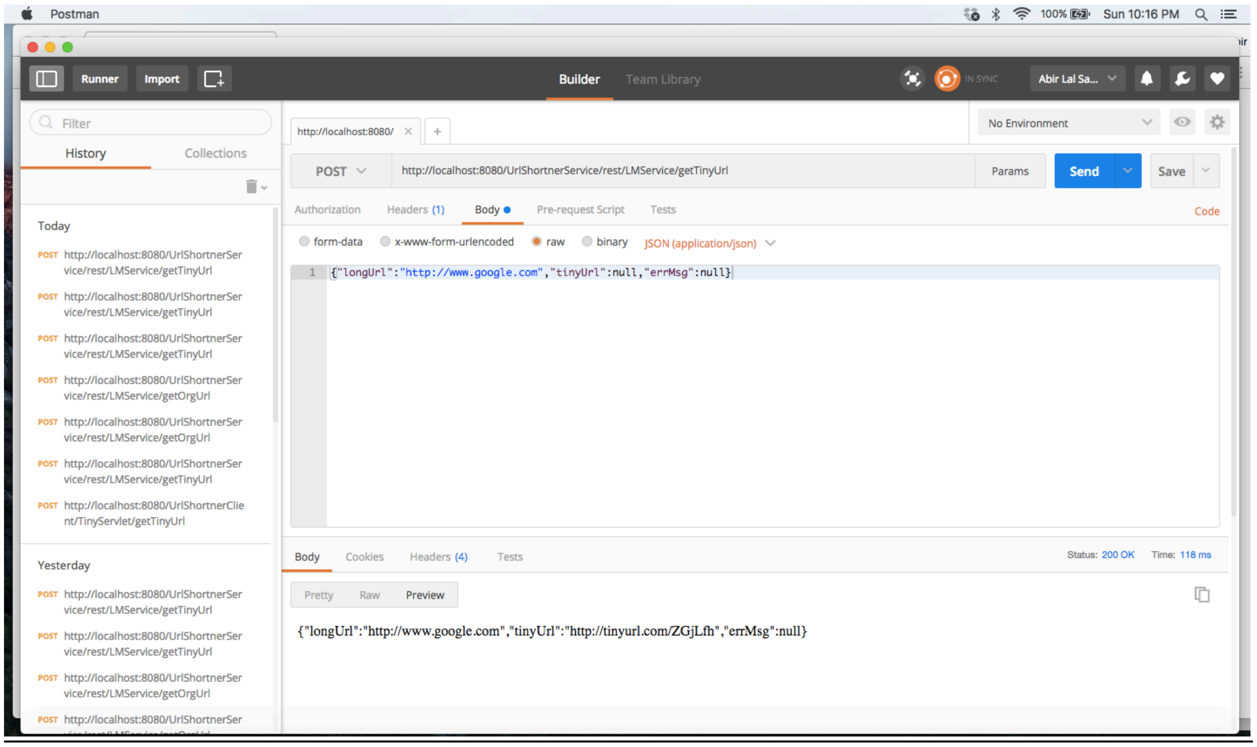


UrlShortnerService



Postman REST client

Webservices Test



Today

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerClient/TinyServlet/getTinyUrl

Yesterday

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl

POST http://localhost:8080/UrlShortnerService

POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl

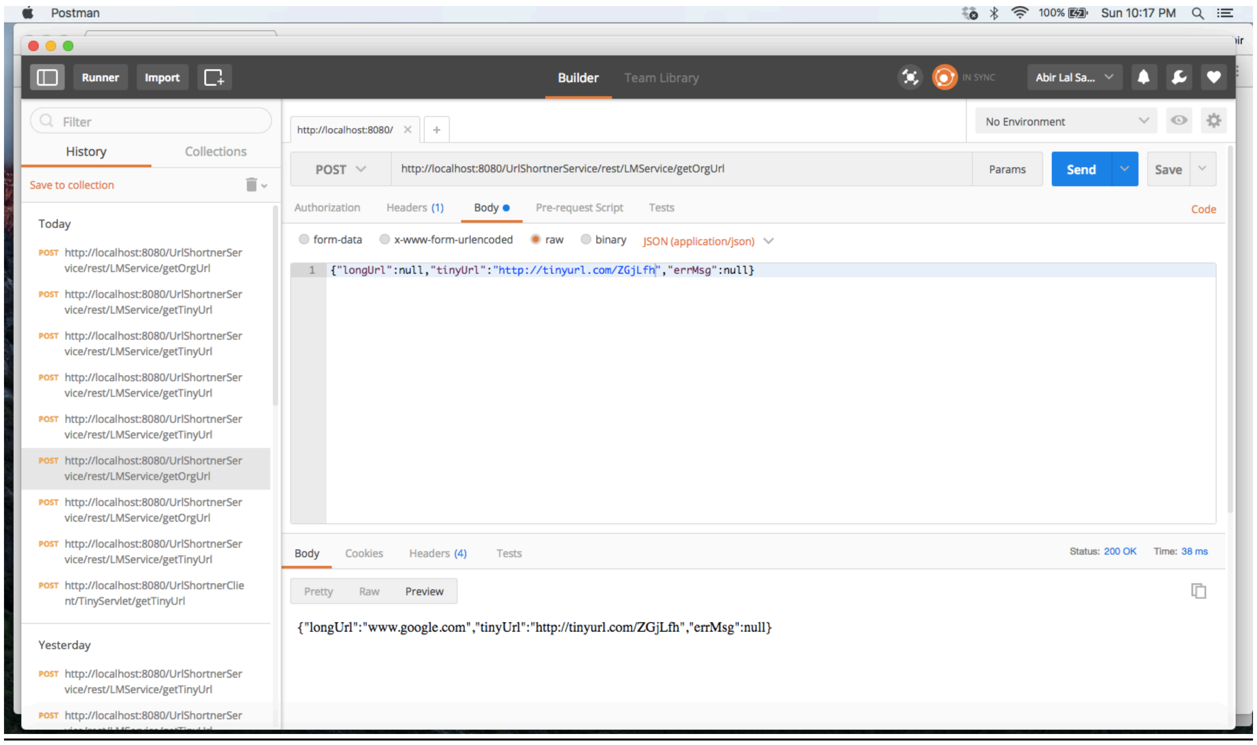
raw — JSON (application/json)

```
{"longUrl":"http://www.google.com","tinyUrl":null,"errMsg":null}
```

Status: 200 OK Time: 118 ms

```
{"longUrl":"http://www.google.com","tinyUrl":"http://tinyurl.com/ZGjLfh","errMsg":null}
```

Endpoint - http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl



No Environment

POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl

Body: form-data · x-www-form-urlencoded · [✓ raw] · binary · JSON (application/json)

```
{"longUrl":null,"tinyUrl":"http://tinyurl.com/ZGjLfh","errMsg":null}
```

Status: 200 OK Time: 38 ms

```
{"longUrl":"www.google.com","tinyUrl":"http://tinyurl.com/ZGjLfh","errMsg":null}
```

Today
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerClient/TinyServlet/getTinyUrl

Yesterday
- POST http://localhost:8080/UrlShortnerService/rest/LMService/getTinyUrl
- POST http://localhost:8080/UrlShortnerSer

Endpoint - http://localhost:8080/UrlShortnerService/rest/LMService/getOrgUrl