Distributed Operating Systems, Project4:

Facebook like Rest API using

Scala, Akka and Spray.

Contents

Team:

Name:	Email:
Prashanth Peddabbu	ppeddabbu@gmail.com
Shivdeep Nutheti	shiva.0891@gmail.com

This has two folders, one is for 'client simulator' and other is for 'server side REST interface'

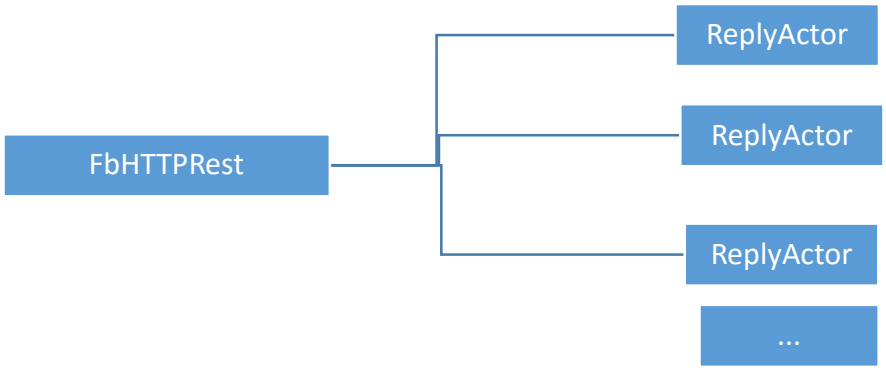
Please follow these steps to execute Server:

1) Go to server folder that has build.sbt, Make sure you have all the Scala, Akka, Spray dependencies that were mentioned in the sbt file.
2) If needed modify the application.config file to different host and port to change the server bind address.
3) Use sbt compile and sbt run the code to execute server.
4) Please follow the exact similar steps as mentioned above to build the client simulator code.

Summary: Server Side (FbHTTPRest)

As mentioned in the class, we have come up with our own version of Facebook REST interface to manage- User Profile, Page, Post, Friend List, Picture, Album APIs (please refer to Table:1 for CRUD operations we supported). As shown in below figure, we have HttpServiceActor – FbHTTPRest who does the job of routing the http requests that comes in from the clients and assigns the http response tasks to its worker actors- ReplyActor. Each Reply actor does the job of sending http response to the clients.

Actors in the Server System:



What Server (FbHTTPRest) performs?

Implemented: Page, Post, Friend List, Profile, Picture, and Album. Below one is the comprehensive list of APIs that we have implemented to simulate the tasks we do in Facebook like environment.

Table: 1

Method	API	Description
PUT	/user	-> create user
GET	/users/{uid}	-> get user
POST	/users/{uid}	-> update user
DELETE	/users/{uid}	-> delete user
PUT	/users/{uid}/page	-> create page
POST	/users/{uid}/pages/{pgid}	-> update page if id is the admin
GET	/users/{uid}/pages/{pgid}	-> get page pgid
GET	/pages/{pgid}	-> get page pgid(for outside people)
POST	/users/{uid}/post	-> post on timeline
GET	/users/{uid}/get_all_posts	-> get all posts of user uid
GET	/users/{uid}/posts/{psid}	-> get post psid
DELETE	/users/{uid}/posts/{psid}	-> delete post psid
POST	/users/{uid}/pages/{pgid}/post	-> post on timeline of the pgid page
GET	/users/{uid}/pages/{pgid}/get_all_posts	-> get all posts of the pgid page
POST	/users/{uid}/like_page	-> like page by user
POST	/users/{uid}/unlike_page	-> unlike page by user
POST	/users/{uid}/add_friend	-> add friend to the user
POST	/users/{uid}/remove_friend	-> remove friend from the user
POST	/users/{uid}/add_profile_pic	-> add profile pic to the user
POST	/users/{uid}/add_album	-> add album to the user

* *url - localhost:7005*

**uid – user id, pgid- page id, psid- post token, albid- album id*

POST	/users/{uid}/albums/{albid}/add_pic	-> To update a pic to existing album
GET	/users/{uid}/albums/{albid}/	-> To Get albid album from the uid user

Our Design model and its Sources:

User Profile:

Each user entity will have fields as shown in the figure. Please refer to table:1 for the CRUD operations we support on user specific tasks.

```
case class User(
  id:              Int,
  username:        Option[String],
  name:            Option[String],
  first_name:      Option[String],
  middle_name:     Option[String],
  last_name:       Option[String],
  email:           Option[String],
  link:            Option[String],
  gender:          Option[String],
  page_list:       List[Int],
  friends_list:    List[Int],
  post_list:       List[Int],
  album_list:      Option[List[Album]],
  profilepicture: Option[Pic] )
```

page_list : List of all liked pages

friend_list : List of all friends

post_list : List of all posts made by this user

profilepicture : It has profile picture of this user

album_list : It has list of albums by this user

User object is mentioned above. CRUD operations for user are supported in this model.

User1 can add user2 as a friend. User2 can unfriend someone in his friends list.

Source: https://developers.facebook.com/docs/graph-api/reference/user

Page:

Each page entity will have fields as shown in the figure. Please refer to table: 1 for the CRUD operations we support on page specific tasks.

```
case class Page(
  id:              Int,
  name:            Option[String],
  admin:           Option[Int],
  count_likes:     Int,
  user_list:       List[Int],
  post_list:       List[Int] )
```

name : name of the page

count_likes : Count of number of users liked that page.

user_list : which has the users list who like this page.

post_list : It has list of posts on that page.

CRUD operations are supported for page. User1 can like some page1. He can unlike page1.

A page can be retrieved even if I am not logged in. (but cannot perform any other operations)

Source: https://developers.facebook.com/docs/graph-api/reference/page

Post:

Each post entity will have fields as shown in the figure. Please refer to table: 1, for the CRUD operations we support on post specific tasks.

message: This field is the content of the post.

postedBy : It stores the userid of the post creator

posted_inpage : It stores the page in which it was posted. (If it is posted on page)

likes : It tracks number of likes of the post.

comments: It tracks comments on that post.

```
case class Post(
  id:            Int,
  message:       String,
  postedBy:      Option[Int],
  posted_inpage: Option[Int],
  likes:         Int,
  comments:      Option[List[String]])
```

User1 can create a post on timeline. User1 can create a post on page1 timeline.

He can like or comment the post.

Source:https://developers.facebook.com/docs/graph-api/reference/v2.5/post/

Friends List:

Please refer to table: 1, for the CRUD operations we support on add_firend and remove_friend (unfriend) specific tasks.

User1 can add user2 as a friend. User2 can unfriend someone in his friends list.

Source: https://developers.facebook.com/docs/graph-api/reference/user/friendlists/

Picture:

Each pic entity will have fields as shown in the figure. Please refer to table: 1, for the CRUD operations we support on post specific tasks.

desc : It is the caption of the Picture

```
case class Pic(
  id:    Int,
  desc:  Option[String],
  image: String)
```

image : In this model, we have take image as string. We can extend to mediatype which requires database for efficiency.

Source: https://developers.facebook.com/docs/graph-api/reference/user/picture/

Album:

Albums is a collection of albums. Album is a collection of pictures.

He can add album1 to his albums.

He can add pic1 to album1.

```
case class Album(
  id:          Int,
  caption:     Option[String],
  pics:        List[Pic])
```

Add profile pic API adds a profile picture. It adds it into "profile_picture" album .It updates profile picture.

Get album API fetches album – id

Source: https://developers.facebook.com/docs/graph-api/reference/user/albums/

Summary: ClientSimulator(FbClient):

The Facebook client simulator takes the input arguments: number of users, number of pages, and number of posts.

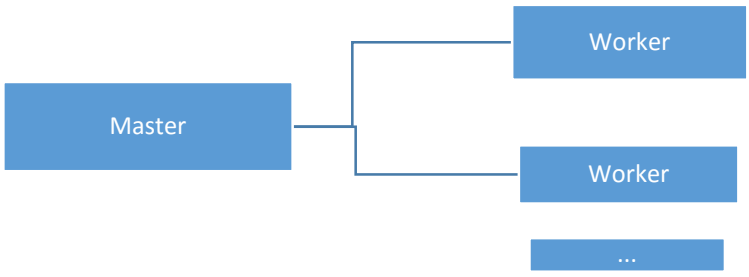
We added methods to simulate create users, create pages, delete users, create posts for time line and create posts on page.
A master is invoked with these parameters. It assigns workers for the job using roundrobinpool.
Each worker takes equal part of the work.
Each worker logs the id of the entity if work was not done.
E.g.: Log: user with id was not posted and response status.

Friendship Network:

Suppose we create 10000 users, the method createFriendshipNetwork creates a network
E.g.: user1 will add friend user 2501, user 5001, user 7501:
Representation of friendship -
(1 =>2501,5001,7501) , (2=> 2502,5002,7502) , ……..(2500=> 5000,7500,10000)

In this method, friends_list of all users are being updated.

Actors in the client system:



Simulation results:

Work	Users Count	Total elapsed time	Workers Count

1. create users	10000 100000 200000 300000 400000	18582 355835 1583075 3227252 6618421	20 20 20 20 20

Work	Pages Count	Total elapsed time (100000 users creation +page creation)	Workers Count
2. Create pages on fixed no of users (count =100000)	10000 100000 200000	372016 +16481 366235 +229871 383093 + 521467	20 20 20

Work	Users Count	Total elapsed time (users creation + friendship)	Workers Count
3. Create users and friendship network on the top of them	10000 100000 200000	15142 + 13155 373187 + 1007934 1583075 + 5392768	20 20 20

Work	Users Count	Pages Count	Posts Count	Total elapsed time (users + pages + posts)	Workers Count
4. Create users, pages and posts. (posts can be on timeline or page)	10000 100000	10000 100000	10000 100000	7232 + 15296+ 20929 359538 + 221509+ 1203169	20 20

Dependencies:

Akka framework, Spray-can, Spray-json, Spray-routing, Spray-http

References:

http://spray.io/documentation/1.2.3/spray-routing/

http://stackoverflow.com/questions/32248210/sending-json-request-with-spray-io-client

http://danielasfregola.com/2015/02/23/how-to-build-a-rest-api-with-spray/

http://alvinalexander.com/scala/scala-akka-futures-example-simple-working

https://developers.facebook.com/docs/graph-api

http://spray.io/documentation/1.2.2/spray-routing/path-directives/pathmatcher-dsl/

Sample API calls

```
create user
http://localhost:7005/user put
{ "id" : 50,
 "username" : "shivdeepn",
 "name" : "shiv",
 "first_name" : "shivdeep",
 "middle_name" : "",
 "last_name" : "nutheti",
 "email": "shiva.0891@gmail.com",
 "link":"linkd",
 "gender": "male",
 "page_list":[],
 "friends_list"  : [],
 "post_list":[]
}

//add friend
http://localhost:7005/users/20/add_friend   post
{ "id" : 50,
 "page_list":[],
 "friends_list"  : [],
 "post_list":[]
}
http://localhost:7005/users/20/remove_friend works similar to the above
api

//create a page
http://localhost:7005/users/20/page  put
{
  "id" : 1234,
  "name": "newpage",
  "count_likes": 0,
  "user_list" : [],
  "post_list" : []
}

//like page
http://localhost:7005/users/20/like_page post
{
  "id" : 1234,
  "name": "newpage",
  "count_likes": 0,
  "user_list" : [],
  "post_list" : []
}
http://localhost:7005/users/20/unlike_page works similar to the above api

//post on page
http://localhost:7005/users/20/pages/1234/post put
{
  "id": 56,
  "message": "String",
```

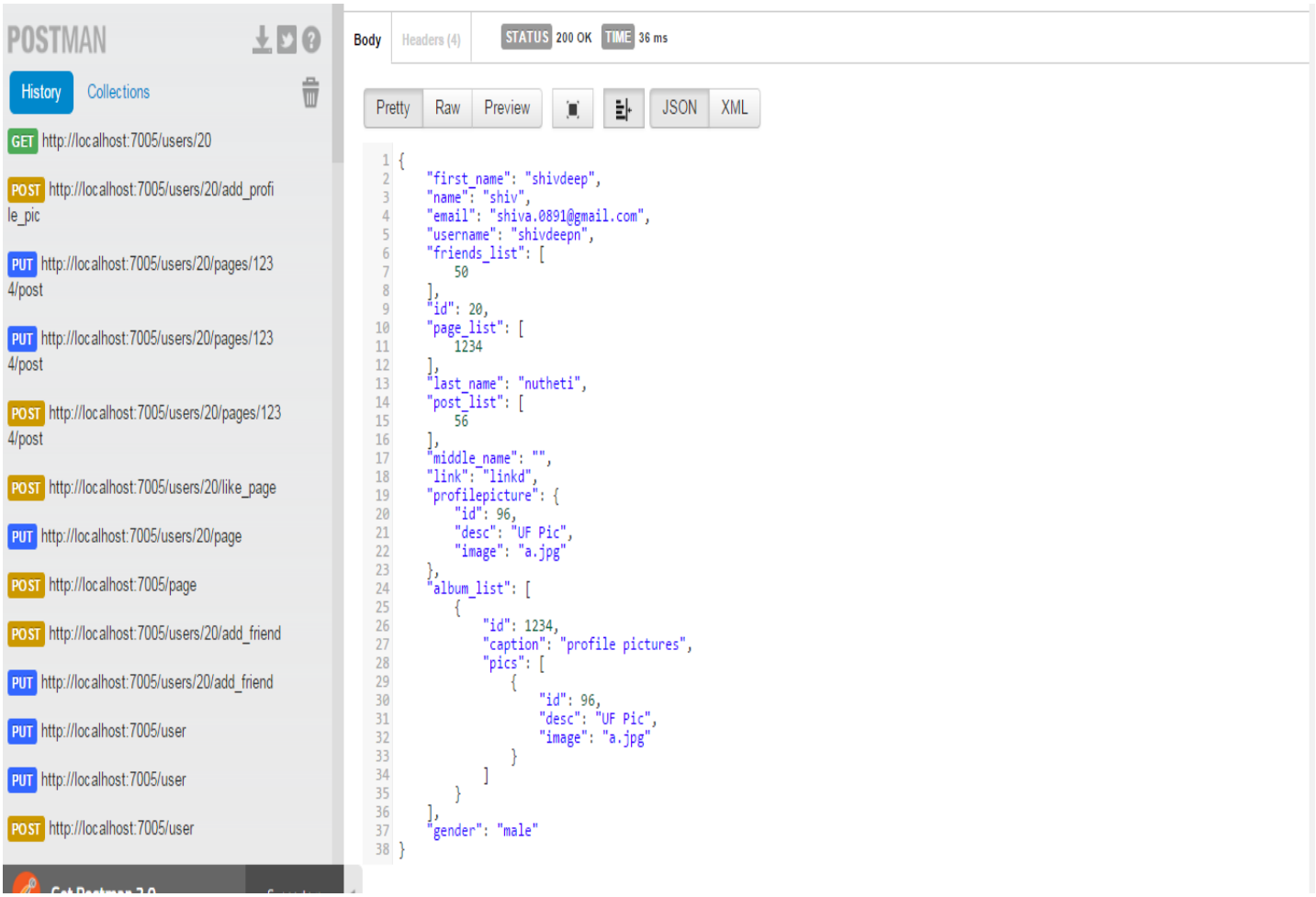
```
  "likes": 0,
  "comments" :[ "comment1", "comment2"]
}

//add profile pic (which goes to profile pictures album)
http://localhost:7005/users/20/add_profile_pic
{
  "id": 96,
  "desc":"UF Pic",
  "image": "a.jpg"
}
//add album
http://localhost:7005/users/20/add_album post
{
  "id": 97,
  "caption" :"my album",
  "pics": [
           { "id": 59,
             "desc" : "description1",
             "image" :"a.jpg"
           },

           { "id": 54,
             "desc" : "description1",
             "image" :"a.jpg"
           }
          ]
}

//add pic to album
http://localhost:7005/users/20/albums/97/add_pic post
{ "id": 70,
  "desc" : "UF pic",
  "image" :"a.jpg"
}
```

Client Simulator Output

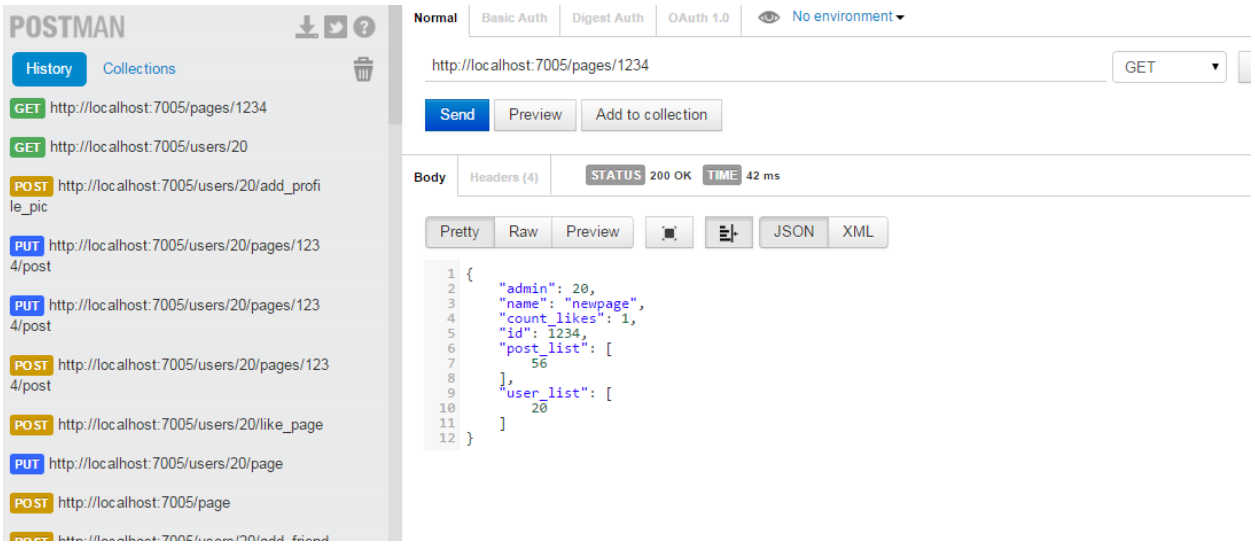
```
-- fb model simulator--
Enter the # of users to simulate>
100000
Enter the # of pages to create out of theses users>
100000
Enter the # of posts to be made by each user on his feed or on pages>
100000
users Creation done
Total elapsed time: 359538
pages Creation done
Total elapsed time: 221509
posts Creation done
Total elapsed time: 1203169
shutting down the simulator
```

Sample screenshots of the POSTMAN client against our Facebook REST API

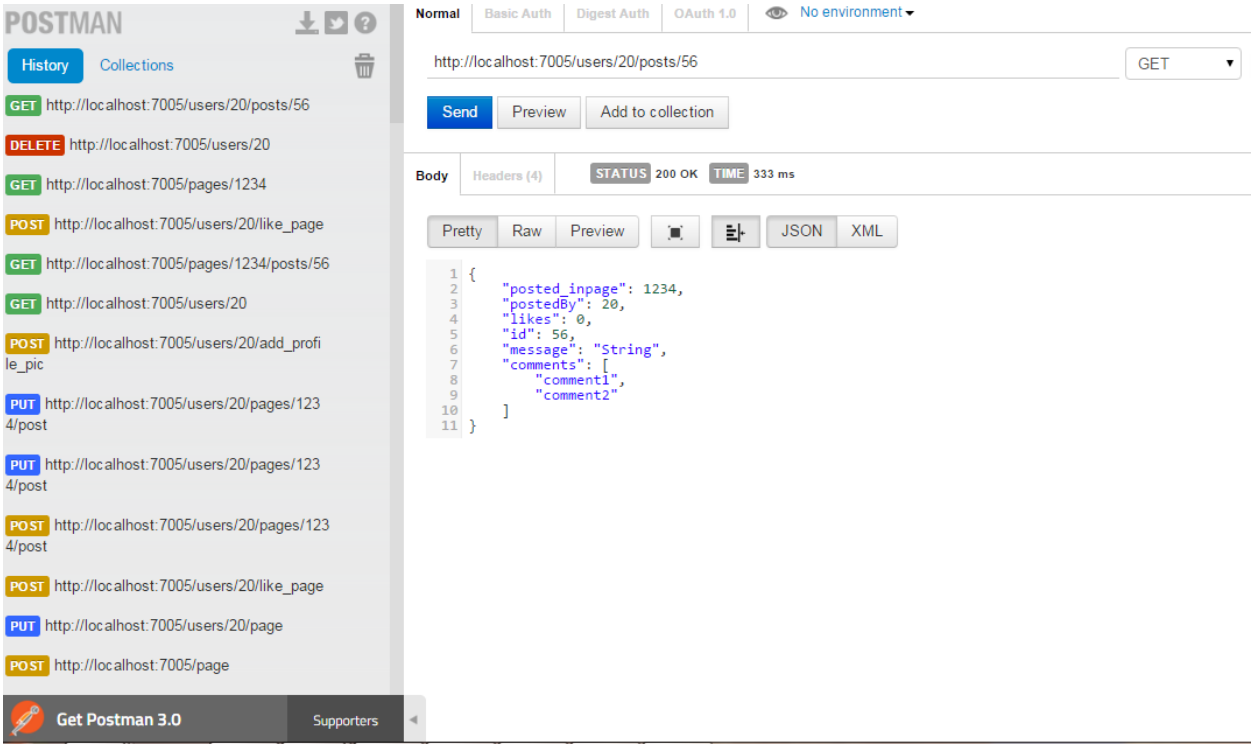
USER JSON data



PAGE JSON data



POST JSON data


POSTMAN
History
Collections
GET http://localhost:7005/users/20/posts/56
DELETE http://localhost:7005/users/20
GET http://localhost:7005/pages/1234
POST http://localhost:7005/users/20/like_page
GET http://localhost:7005/pages/1234/posts/56
GET http://localhost:7005/users/20
POST http://localhost:7005/users/20/add_profile_pic
PUT http://localhost:7005/users/20/pages/1234/post
PUT http://localhost:7005/users/20/pages/1234/post
POST http://localhost:7005/users/20/pages/1234/post
POST http://localhost:7005/users/20/like_page
PUT http://localhost:7005/users/20/page
POST http://localhost:7005/page
Get Postman 3.0
Supporters
Normal
Basic Auth
Digest Auth
OAuth 1.0
No environment
http://localhost:7005/users/20/posts/56
GET
Send
Preview
Add to collection
Body
Headers (4)
STATUS 200 OK
TIME 333 ms
Pretty
Raw
Preview
JSON
XML
{
"posted_inpage": 1234,
"postedBy": 20,
"likes": 0,
"id": 56,
"message": "String",
"comments": [
"comment1",
"comment2"
]
}