Backend Solution Documentation (Question 2)

Database Schema

SQLite database is used for sake of simplicity and portability.
Database – social.db

Tables

users(userid, username): userid is the PK

	userid	username
	Filter	Filter
1	1	Dave
2	2	Leo
3	3	Shawn
4	4	Robert

posts(postid, userid, title, content): postid is the PK, userid is FK reference of users(userid)

	postid	userid	title	content
	Filter	Filter	Filter	Filter
1	1	1	post1 by user1	content 1
2	2	2	post1 by user2	content 2
3	3	2	post2 by user2	content 3
4	4	4	first post by Robert	I really completed it
5	5	4	second post by Robert	Its my 2nd post

comments(commentid, postid, userid, comment): commentid is the PK, postid is FK reference of posts(postid), userid is FK reference of users(userid)

	commentid	postid	userid	comment
	Filter	Filter	Filter	Filter
1	3	1	2	comment1
2	4	1	1	own post comment
3	5	1	4	this post is nice
4	6	1	4	another comment because this post i...
5	7	2	4	comment spree
6	8	3	4	I am done
7	9	5	1	comment on some post

postlikes(likeid, postid, userid): likeid is the PK, postid is FK reference of posts(postid), userid is FK reference of users(userid)

	likeid	postid	userid
	Filter	Filter	Filter
1	1	1	1
2	2	2	1
3	3	1	2
4	4	3	1
5	5	3	3
6	6	5	1

commentlikes(likeid, commentid, userid): likeid is the PK, commentid is FK reference of comments(commentid), userid is FK reference of users(userid)

	likeid	commentid	userid
	Filter	Filter	Filter
1	1	3	2
2	2	3	3
3	3	4	3
4	4	3	1
5	5	3	4
6	6	6	3

API Reference

POST **/posts**: to create a post

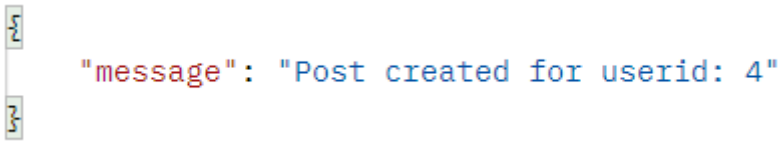
Takes “userid”, “title” and “content” as request body. Post will be created for the given userid.

Body example:

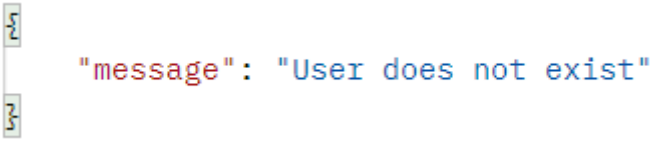
```
{
    "userid": 4,
    "title": "second post by Robert",
    "content": "Its my 2nd post"
}
```

Responses

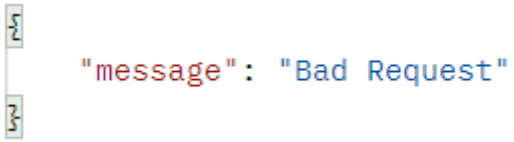
On successful creation of a post



For an invalid user



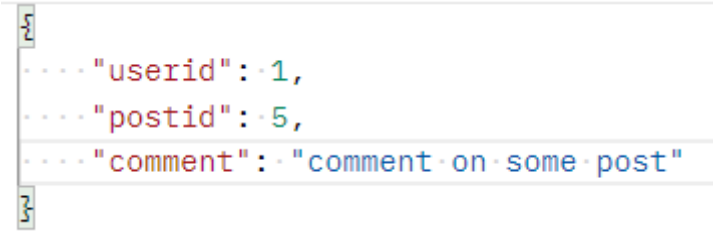
Missing required body



POST **/comments**: to comment on a post

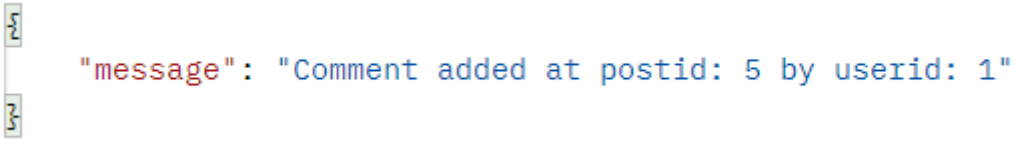
Takes “postid”, “userid” and “comment” as request body. Comment will be added under the given postid by the given userid.

Body example:

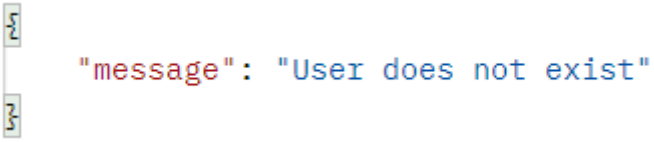


Responses

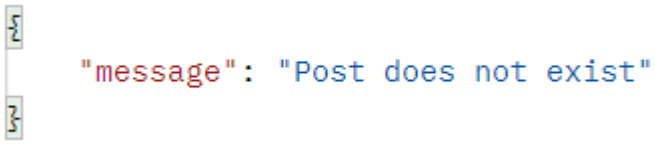
On successful comment



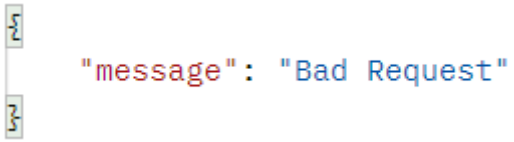
For an invalid user



For an invalid post



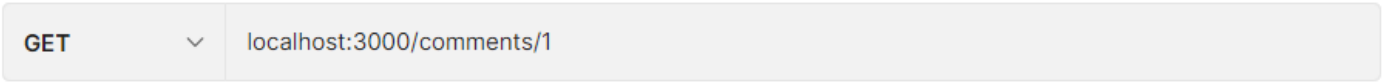
Missing required body



GET **/comments/<userid>**: fetch the latest list of users commenting on posts of userid

Takes “userid” as a request parameter. If the user itself comments on its own post then that user will not be present in the list.

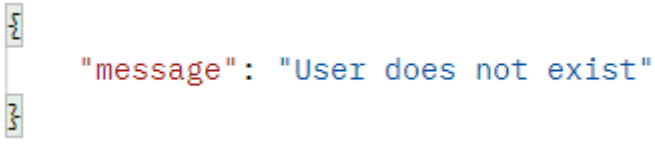
Request URL example:



Responses

On successful fetch

For an invalid user



Missing request parameter

```
{
    "message": "Bad Request"
}
```


POST **/likePosts**: to like a post

Takes “postid”, “userid” request body. Post with given postid will be liked by the given userid. A user cannot repeatedly like the same post.

Body example:


```
{
    "userid": 1,
    "postid": 5
}
```


Responses

On successfully liking a post


```
{
    "message": "postid: 5 liked by userid: 1"
}
```


Again liking the same post


```
{
    "message": "Post has already been liked by user"
}
```


For an invalid user


```
{
    "message": "User does not exist"
}
```


For an invalid post


```
{
    "message": "Post does not exist"
}
```

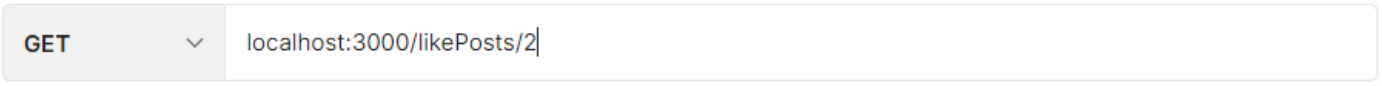

Missing required body

```
{
    "message": "Bad Request"
}
```


GET **/likePosts/<userid>**: fetch the latest list of users liking the posts of userid

Takes “userid” as a request parameter. If the user itself likes its own post then that user will not be present in the list.

Request URL example:



Responses

On successful fetch


```
{
    "users": [
        {
            "userid": 1,
            "username": "Dave"
        },
        {
            "userid": 3,
            "username": "Shawn"
        }
    ]
}
```


For an invalid user


```
{
    "message": "User does not exist"
}
```


Missing request parameter


```
{
    "message": "Bad Request"
}
```

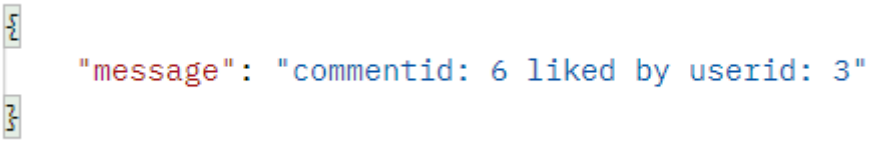

POST **/likeComments**: to like a comment

Takes “commentid”, “userid” request body. Comment with given commentid will be liked by the given userid. A user cannot repeatedly like the same comment.

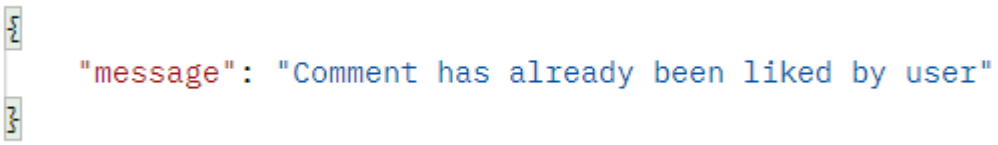
Body example:

Responses

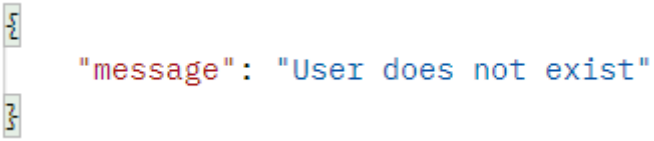
On successfully liking a comment



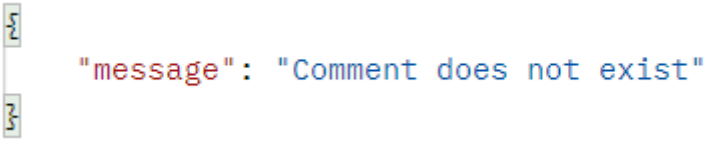
Again liking the same comment



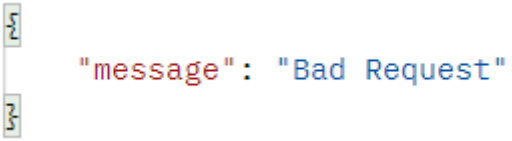
For an invalid user



For an invalid comment



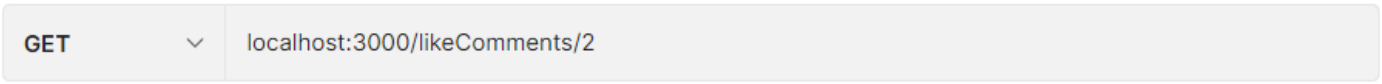
Missing required body



GET **/likeComments/<userid>**: fetch the latest list of users liking the comments by userid on any post.

Takes “userid” as a request parameter. If the user itself likes its own comments then that user will not be present in the list.

Request URL example:



Responses

On successful fetch


```
{
    "users": [
        {
            "userid": 3,
            "username": "Shawn"
        },
        {
            "userid": 1,
            "username": "Dave"
        },
        {
            "userid": 4,
            "username": "Robert"
        }
    ]
}
```


For an invalid user


```
{
    "message": "User does not exist"
}
```


Missing request parameter


```
{
    "message": "Bad Request"
}
```


Note – All the above scenarios are tested by serving from localhost and using Postman for the API calls

Features

- All API endpoints are made available as per the problem statement
- Activities are idempotent. A user cannot like the same post/comment on a post.
- The fetch APIs to list users liking posts or comments on posts by a user will not have that same user on the list even if it likes its own post or comments on post. Although this information will remain in the DB.
- The above mentioned feature is developed to simulate the behaviour that a user should not receive notification if it likes its own post or comments on post.

Installation

- Install NodeJS.
- Execute `npm install` inside the parent directory to install the required node modules.
- Execute `node app.js` to start the server.