

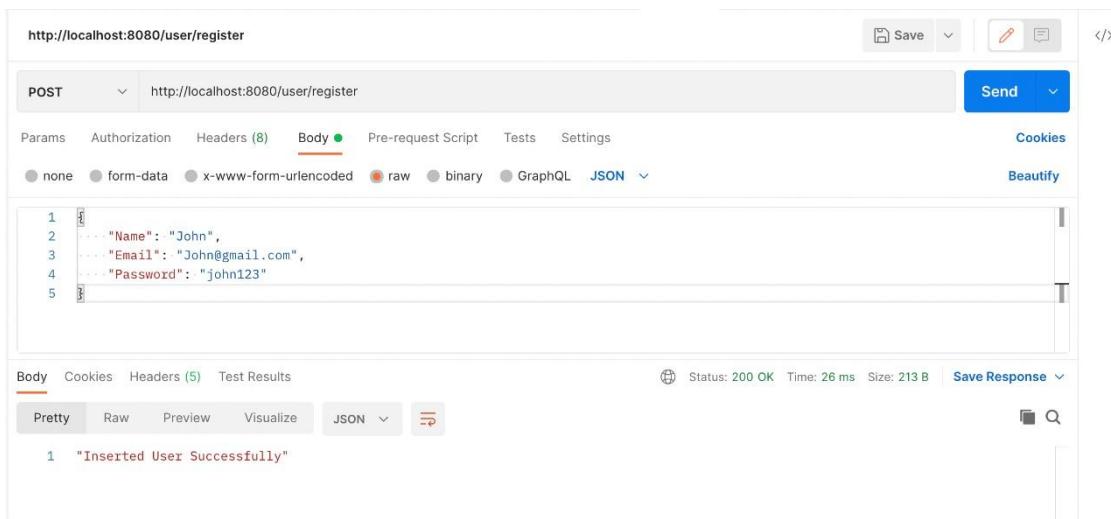
SPRINT 4

BACKEND API DOCUMENTATION

USER APIs:

1. insertuser

Through this API the new user data is inserted into the user table.



http://localhost:8080/user/register

POST http://localhost:8080/user/register

Body (8) JSON

```
1 {"Name": "John",
2 "Email": "John@gmail.com",
3 "Password": "john123"}  
4  
5
```

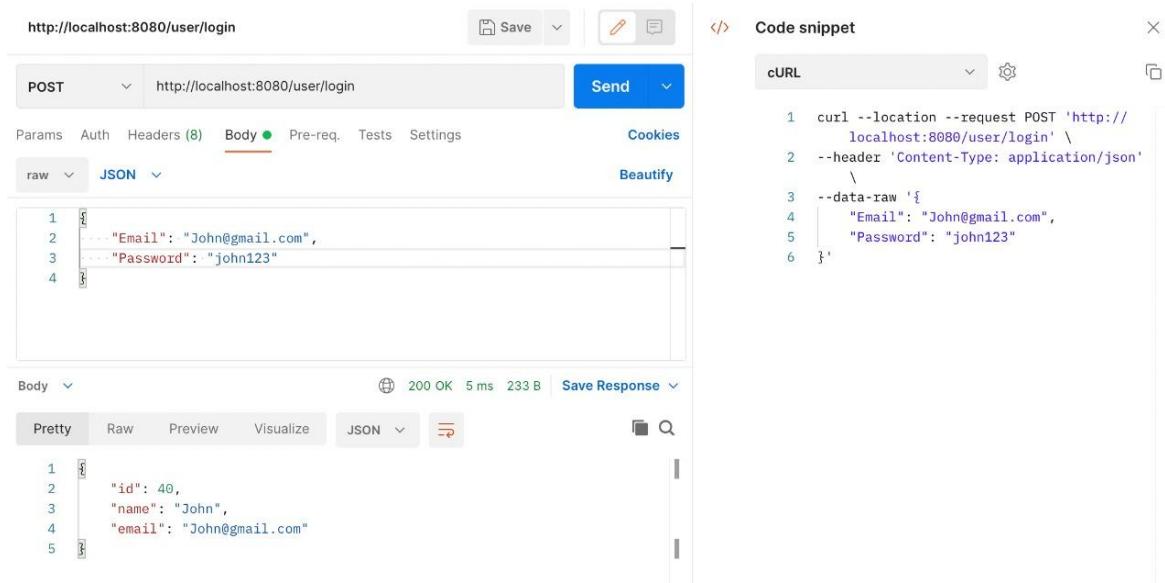
Status: 200 OK Time: 26 ms Size: 213 B Save Response

Pretty Raw Preview Visualize JSON

1 "Inserted User Successfully"

2. loginuser

The data from user like the email and password is taken and compared with the database and user is logged into the application



http://localhost:8080/user/login

POST http://localhost:8080/user/login

Body (8) JSON

```
1 {"Email": "John@gmail.com",
2 "Password": "john123"}  
3  
4
```

Status: 200 OK 5 ms 233 B Save Response

Pretty Raw Preview Visualize JSON

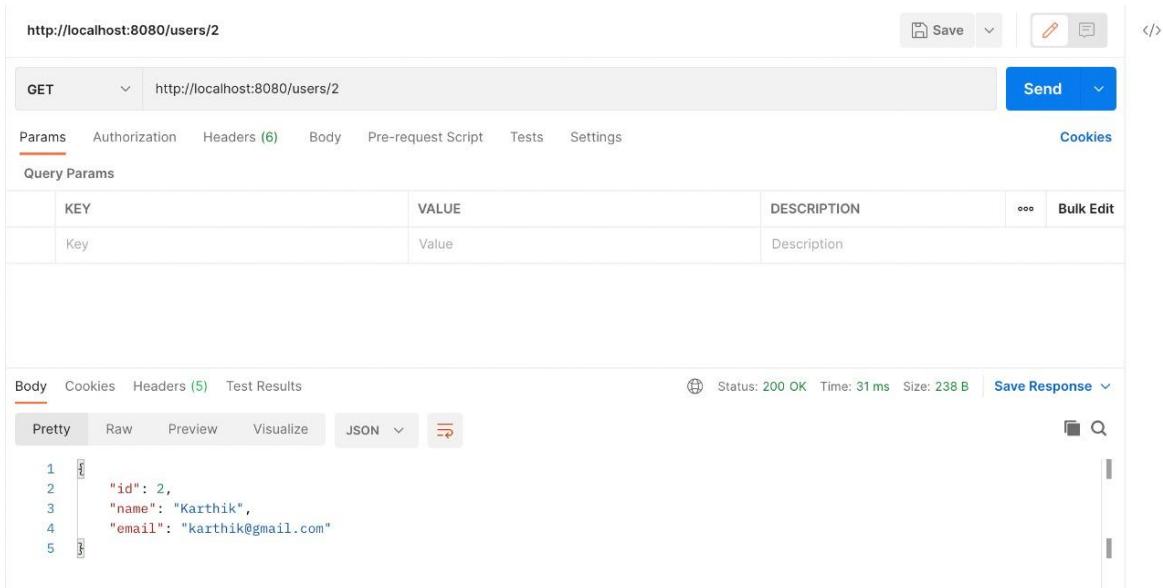
Code snippet

```
cURL  
1 curl --location --request POST 'http://localhost:8080/user/login' \
2 --header 'Content-Type: application/json' \
3 --data-raw '{'
4   "Email": "John@gmail.com",
5   "Password": "john123"
6 }'
```

1 {"id": 40,
2 "name": "John",
3 "email": "John@gmail.com"}
4
5

3. retrieveuser

This API is used to check if the given user exists in the database and it takes the user id and if the user already exists the user data is returned and displayed.



The screenshot shows a Postman request for `http://localhost:8080/users/2`. The method is set to `GET`. The response status is `200 OK` with a time of `31 ms` and a size of `238 B`. The response body is a JSON object:

```
1 "id": 2,
2 "name": "Karthik",
3 "email": "karthik@gmail.com"
```

4. getallusers

This API does not take any parameters as an input but retrieves all the users present in the database which is used to display a drop down list in the front end

http://localhost:8080/users/all

GET http://localhost:8080/users/all

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 []
2   "Teja",
3   "Karthik",
4   "Karthik1",
5   "",
6   "Vikranth",
7   "Anusha",
8   "Anusha",
9   "Anusha",
10  "Karthik1",
11  "Karthik1",
12  "Karthik1",
13  "Vikranth",
14  "uday",
15  "rekh",
16  "Vikranth",
17  "Vikranth",
18  "Vikranth",
19  "uday",
20  "udav".

```

Status: 200 OK Time: 33 ms Size: 583 B Save Response

5. updateuserpassword

This API is used when a user wants to change the current password. It takes the userid and the new password and is accordingly updated in the database.

http://localhost:8080/user/passwordupdate

PUT http://localhost:8080/user/passwordupdate

Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies Beautify

raw JSON

```

1 {
2   "userid" : "1",
3   "password": "newpassword"
4 }

```

Body 200 OK 9 ms 216 B Save Response

Pretty Raw Preview Visualize JSON

```

1 "Password updated Successfully"

```

Code snippet

cURL

```

1 curl --location --request PUT 'http://localhost:8080/user/passwordupdate' \
2 --header 'Content-Type: application/json' \
3 --data-raw '{
4   "userid" : "1",
5   "password": "newpassword"
6 }'

```

Expense APIs:

1. insertexpense

InsertExpense is an API where a new expense is inserted into the expense database.

The screenshot shows the Postman interface. The URL is `http://localhost:8080/expense`. A POST request is being made. The Body tab is selected, showing the following JSON payload:

```
1 {
2   "name": "exp1",
3   "spent_on": "chikfila",
4   "category": "Food",
5   "amount": "23.0",
6   "userid": "3"
7 }
```

The response status is 200 OK, and the response body is "Expense created Successfully".

2. getexpense

When the name, category and description of the expense is given the expense amount is retrieved and returned to the frontend.

The screenshot shows the Postman interface. The URL is `http://localhost:8080/expense/get`. A GET request is being made. The Body tab is selected, showing the following JSON payload:

```
1 {
2   "name": "springbreak",
3   "category": "food",
4   "spent_on": "burgers"
5 }
```

The response status is 200 OK, and the response body is a JSON object with fields id, userid, name, category, spent_on, description, and amount.

3. updateexpense

updateexpense takes the name and amount to be updated and accordingly maps and updates the corresponding amount of the expense.

The screenshot shows the Postman interface with the following details:

- Request URL:** http://localhost:8080/expense/updateincome
- Method:** PUT
- Body (JSON):**

```

1 {
2   "amount": "89",
3   "name": "springbreak"
4 }
```
- Response Status:** 200 OK
- Response Body:**

```
1 "Expense updation Successfully"
```
- Code snippet (cURL):**

```

1 curl --location --request PUT 'http://
localhost:8080/expense/update' \
2 --header 'Content-Type: application/json'
3 --data-raw '{
4   "amount": "89",
5   "name": "springbreak"
6 }'
```

Income APIs:

1. insertincome

A new income details of the user inserted into the income relation of the database.

The screenshot shows the Postman interface with the following details:

- Request URL:** http://localhost:8080/expense/insertincome
- Method:** POST
- Body (JSON):**

```

1 {
2   "income_source": "payroll",
3   "amount": "1000",
4   "user_id": "12",
5   "description": "",
6   "timestamp": "04-20-22"
7 }
```
- Response Status:** 200 OK
- Response Body:**

```
1 "Income record created Successfully"
```
- Code snippet (cURL):**

```

1 curl --location --request POST 'http://
localhost:8080/expense/insertincome' \
2 --header 'Content-Type: application/json'
3 --data-raw '{
4   "income_source": "payroll",
5   "amount": "1000",
6   "user_id": "12",
7   "description": "",
8   "timestamp": "04-20-22"
9 }'
```

2. updateincome

Whenever the user has an update on that particular income this API is called. It takes the income id and the amount to be updated and respectively updates the income amount in the database.

http://localhost:8080/expense/updateincome

PUT http://localhost:8080/expense/updateincome

Params Auth Headers (8) Body **JSON** Pre-req. Tests Settings Cookies

```

1 {
2   "amount": "12",
3   "ID": "1"
4 }

```

Body **JSON** 200 OK 26 ms 208 B Save Response

Pretty Raw Preview Visualize JSON

1 "Updated Income record"

3. deleteincome

When data about the corresponding income is no longer needed this API is invoked to delete the details of that income.

http://localhost:8080/expense/updateincome

DELETE http://localhost:8080/expense/deleteincome/1

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body **JSON** 200 OK 9 ms 209 B Save Response

Pretty Raw Preview Visualize JSON

1 "Income record deleted!"

Payment

1. insertpaymentsplit

This API is invoked when a new group expense is created and it is to be split between a set of borrowers. These split expenses are inserted into the payment_split database with each new record for every borrower.

Code snippet:

```

1 curl --location --request POST 'http://localhost:8080/expense/insertpaymentsplit' \
2 --header 'Content-Type: application/json' \
3 --data-raw '{
4   "borrowers": "teja,anusha",
5   "amount": "90",
6   "user_id": "12",
7   "username": "Eshwar",
8   "description": "starbucks",
9   "timestamp": "04-20-22"
10 }'

```

2. deletepaymentsplit

Once the amount debt has been cleared the data related to this expense will be deleted using this deletepaymentsplit API

Code snippet:

```

1 curl --location --request DELETE 'http://localhost:8080/expense/deletepaymentsplit/2'

```

3. useramountowed

This is one of the most important functions of the application. This is an API where an aggregate on the data is performed to know how much amount is owed to the given user by all the borrowers.

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:8080/expense/deletepaymentsplit/2`. Below it, a new request is being prepared with the URL `http://localhost:8080/expense/useramountowed/ab`. The method is set to `GET`. The `Headers` tab is selected, showing `(6)`. The `Body` tab is selected, showing `Query Params`. A table for query parameters is present with one row: `Key`, `Value`, and `Description`. The `Body` tab also displays the response status as `200 OK` with a response time of `6 ms` and a size of `186 B`. The response body contains the number `1 40`.

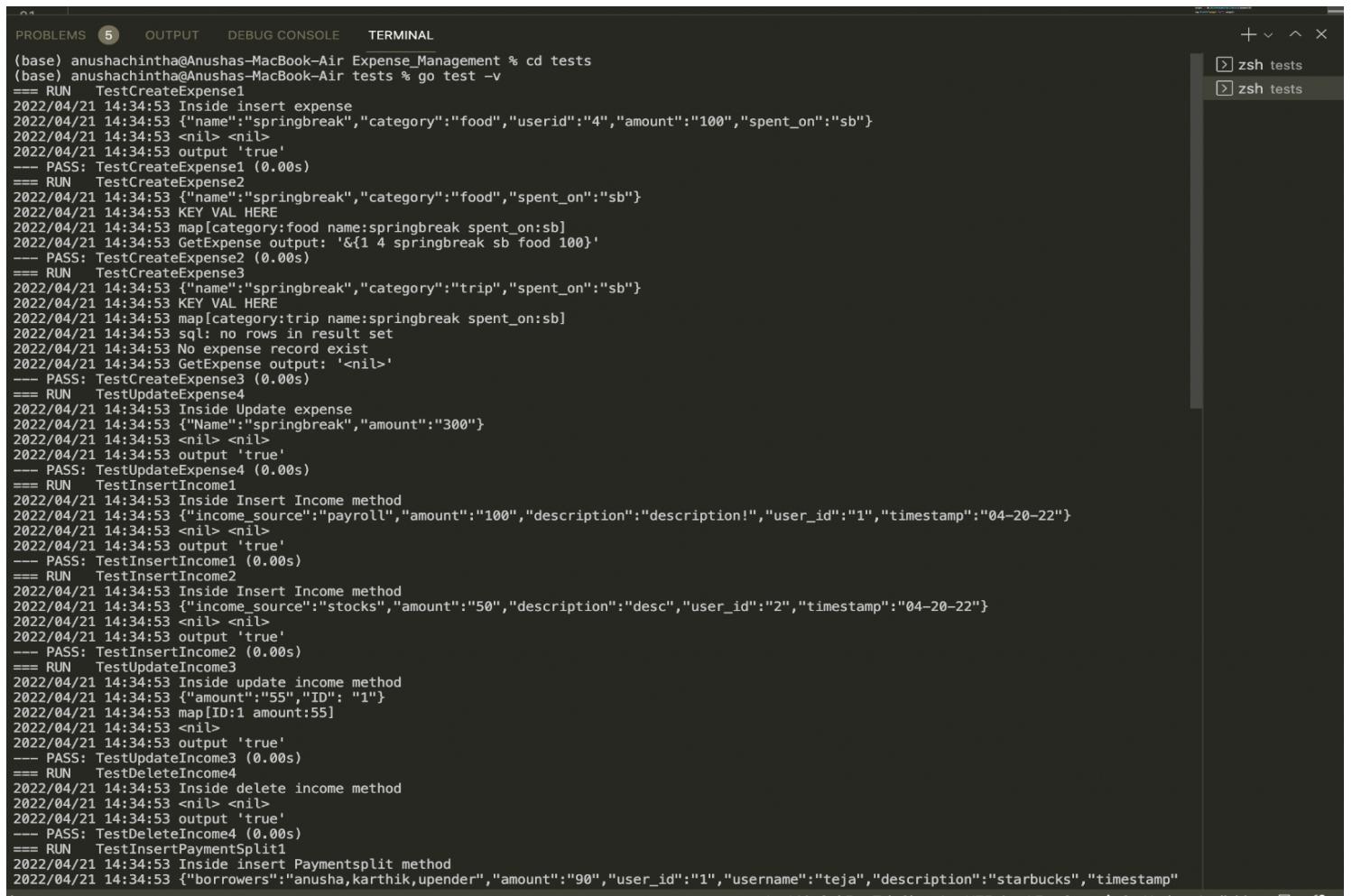
4. userowes

This API's main function is to identify how much amount that particular user owes in total to any other users. This API also performs aggregate operation on the data in the payment_split database and generates the total the user owes to all the other users.

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:8080/expense/user/owes/teja`. Below it, a new request is being prepared with the URL `http://localhost:8080/expense/user/owes/teja`. The method is set to `GET`. The `Headers` tab is selected, showing `(6)`. The `Body` tab is selected, showing `Query Params`. A table for query parameters is present with one row: `Key`, `Value`, and `Description`. The `Body` tab also displays the response status as `200 OK` with a response time of `5 ms` and a size of `188 B`. The response body contains the number `1 52.5`.

BACKEND UNIT TEST RUNNING

After changing the path to the tests folder the command \$go test -v is executed to run all the backed unit tests.



The screenshot shows a terminal window with the following details:

- Header: PROBLEMS (5), OUTPUT, DEBUG CONSOLE, TERMINAL.
- Right sidebar: zsh tests (x2).
- Output content:

```
(base) anushachinth@Anushas-MacBook-Air Expense_Management % cd tests
(base) anushachinth@Anushas-MacBook-Air tests % go test -v
==== RUN TestCreateExpense1
2022/04/21 14:34:53 Inside insert expense
2022/04/21 14:34:53 {"name":"springbreak","category":"food","userid":"4","amount":"100","spent_on":"sb"}
2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestCreateExpense1 (0.00s)
==== RUN TestCreateExpense2
2022/04/21 14:34:53 {"name":"springbreak","category":"food","spent_on":"sb"}
2022/04/21 14:34:53 KEY VAL HERE
2022/04/21 14:34:53 map[category:food name:springbreak spent_on:sb]
2022/04/21 14:34:53 GetExpense output: '&{1 4 springbreak sb food 100}'
--- PASS: TestCreateExpense2 (0.00s)
==== RUN TestCreateExpense3
2022/04/21 14:34:53 {"name":"springbreak","category":"trip","spent_on":"sb"}
2022/04/21 14:34:53 KEY VAL HERE
2022/04/21 14:34:53 map[category:trip name:springbreak spent_on:sb]
2022/04/21 14:34:53 sql: no rows in result set
2022/04/21 14:34:53 No expense record exist
2022/04/21 14:34:53 GetExpense output: '<nil>'
--- PASS: TestCreateExpense3 (0.00s)
==== RUN TestUpdateExpense4
2022/04/21 14:34:53 Inside Update expense
2022/04/21 14:34:53 {"Name":"springbreak","amount":"300"}
2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestUpdateExpense4 (0.00s)
==== RUN TestInsertIncome1
2022/04/21 14:34:53 Inside Insert Income method
2022/04/21 14:34:53 {"income_source":"payroll","amount":"100","description":"description!","user_id":"1","timestamp":"04-20-22"}
2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestInsertIncome1 (0.00s)
==== RUN TestInsertIncome2
2022/04/21 14:34:53 Inside Insert Income method
2022/04/21 14:34:53 {"income_source":"stocks","amount":"50","description":"desc","user_id":"2","timestamp":"04-20-22"}
2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestInsertIncome2 (0.00s)
==== RUN TestUpdateIncome3
2022/04/21 14:34:53 Inside update income method
2022/04/21 14:34:53 {"amount":"55","ID": "1"}
2022/04/21 14:34:53 map[ID:1 amount:55]
2022/04/21 14:34:53 <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestUpdateIncome3 (0.00s)
==== RUN TestDeleteIncome4
2022/04/21 14:34:53 Inside delete income method
2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:53 output 'true'
--- PASS: TestDeleteIncome4 (0.00s)
==== RUN TestInsertPaymentsplit1
2022/04/21 14:34:53 Inside insert Paymentsplit method
2022/04/21 14:34:53 {"borrowers":"anusha,karthik,upender","amount":"90","user_id":"1","username":"teja","description":"starbucks","timestamp"
```

```

2022/04/21 14:34:53 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestInsertPaymentsSplit1 (0.00s)
== RUN TestInsertPaymentsSplit2
2022/04/21 Inside insert Paymentsplit method
2022/04/21 {"borrowers":"anusha,upender","amount":"40","user_id":"2","username":"karthik","description":"dinner","timestamp":"04-14-22"}
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestInsertPaymentsSplit2 (0.00s)
== RUN TestInsertPaymentsSplit3
2022/04/21 Inside insert Paymentsplit method
2022/04/21 {"borrowers":"karthik,upender","amount":"30","user_id":"1","username":"teja","description":"cab","timestamp":"04-10-22"}
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestInsertPaymentsSplit3 (0.00s)
== RUN TestAmountUserOwed
2022/04/21 14:34:54 in amount function
2022/04/21 14:34:54 87.5
2022/04/21 14:34:54 87.5
--- PASS: TestAmountUserOwed (0.00s)
== RUN TestAmountUserOwes
2022/04/21 14:34:54 32.5
--- PASS: TestAmountUserOwes (0.00s)
== RUN TestDeletePaymentSplit
2022/04/21 14:34:54 Inside delete paymentsplit method
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestDeletePaymentSplit (0.00s)
== RUN TestCreateEvent0
2022/04/21 14:34:54 {"Email":"karthik@gmail.com","Password":"12345"}
2022/04/21 14:34:54 output 'true'
--- PASS: TestCreateEvent0 (0.00s)
== RUN TestCreateEvent1
2022/04/21 14:34:54 entered
2022/04/21 14:34:54 {"Email":"karthik@gmail.com","Password":"12345"}
2022/04/21 14:34:54 entered e'karthik@gmail.com'
2022/04/21 14:34:54 entered p'12345'
2022/04/21 14:34:54 output '&{1 karthik@gmail.com 12345}'
--- PASS: TestCreateEvent1 (0.00s)
== RUN TestCreateEvent2
2022/04/21 14:34:54 entered
2022/04/21 14:34:54 {"Email":"anusha@gmail.com","Password":"anushachinthia"}
2022/04/21 14:34:54 entered e'anusha@gmail.com'
2022/04/21 14:34:54 entered p'anushachinthia'
2022/04/21 14:34:54 User doesn't exist!
2022/04/21 14:34:54 output '<nil>'
```

Ln 143, Col 5 Tab Size: 4 UTF-8 LF Go ▲ Go Update Available ⌂ ⌂

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2022/04/21 14:34:54 {"borrowers":"karthik,upender","amount":"30","user_id":"1","username":"teja","description":"cab","timestamp":"04-10-22"}
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestInsertPaymentsSplit3 (0.00s)
== RUN TestAmountUserOwed
2022/04/21 14:34:54 in amount function
2022/04/21 14:34:54 87.5
2022/04/21 14:34:54 87.5
--- PASS: TestAmountUserOwes (0.00s)
== RUN TestDeletePaymentSplit
2022/04/21 14:34:54 Inside delete paymentsplit method
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestDeletePaymentSplit (0.00s)
== RUN TestCreateEvent0
2022/04/21 14:34:54 {"Email":"karthik@gmail.com","Password":"12345"}
2022/04/21 14:34:54 output 'true'
--- PASS: TestCreateEvent0 (0.00s)
== RUN TestCreateEvent1
2022/04/21 14:34:54 entered
2022/04/21 14:34:54 {"Email":"karthik@gmail.com","Password":"12345"}
2022/04/21 14:34:54 entered e'karthik@gmail.com'
2022/04/21 14:34:54 entered p'12345'
2022/04/21 14:34:54 output '&{1 karthik@gmail.com 12345}'
--- PASS: TestCreateEvent1 (0.00s)
== RUN TestCreateEvent2
2022/04/21 14:34:54 entered
2022/04/21 14:34:54 {"Email":"anusha@gmail.com","Password":"anushachinthia"}
2022/04/21 14:34:54 entered e'anusha@gmail.com'
2022/04/21 14:34:54 entered p'anushachinthia'
2022/04/21 14:34:54 User doesn't exist!
2022/04/21 14:34:54 output '<nil>'
--- PASS: TestCreateEvent2 (0.00s)
== RUN TestCreateEvent5
2022/04/21 14:34:54 {"Email":"anusha@gmail.com","Password":"anushachinthia"}
2022/04/21 14:34:54 output 'true'
--- PASS: TestCreateEvent5 (0.00s)
== RUN TestCreateEvent3
outside
2022/04/21 14:34:54 {"Email":"anusha@gmail.com","Password":"expensemangement"}
In update password
2022/04/21 14:34:54 <nil> <nil>
2022/04/21 14:34:54 Output 'true'
--- PASS: TestCreateEvent3 (0.00s)
== RUN TestCreateEvent6
2022/04/21 14:34:54 Output '[ ]'
--- PASS: TestCreateEvent6 (0.00s)
PASS
ok      expenseManagement/tests 0.434s
(base) anushachinthia@Anushas-MacBook-Air tests %
```