



Paas project

Produced by G.O.4 Team

Purpose and summary	3
----------------------------------	---

Data base

• User.....	4
• Sites.....	6

How to work

• Signup and Activation Account.....	7
• Add site to user Account.....	7
• Get and confirm code.....	7
• Delete Account with Email.....	7
• Change Device.....	7
• Change Number.....	7

Requests and Responses

Mobile Application

• User signup.....	8
• User login.....	9
• Active user Account.....	10
• Add Site to user Account.....	11
• Get the password.....	12
• Send recovery link.....	13
• Get user Account information.....	14
• Delete Account.....	15
• Add Email to Account (send link).....	16
• Active Email.....	17
• Show QR Code (for change Device Id).....	18
• Get QRCode for change DeviceID.....	19
• Get link for change Number.....	20
• Change Number with link.....	21

Site

• Recovery password.....	22
• Site Registration	23
• Active site Account.....	24
• Site login.....	25
• Add users to site database.....	26
• Confirm code.....	27
• Send recovery link for sites.....	28
• Sites Account renewal.....	29
• Add previous users.....	30
• Dashboard informations.....	31
• Get user code.....	32
• Send delete Account link to Email.....	33
• Delete Account by link.....	34

Purpose and summary

The purpose of this program is to use the second password to increase the security of logging in the sites.

In this way, after logging in, you log in to your paas account in the application and receive the second password from the box of the site, and enter the second password in the password request section.

All passwords are stored hashed on the site so that passwords will not be accessible if anonymous people access them.

DateBase

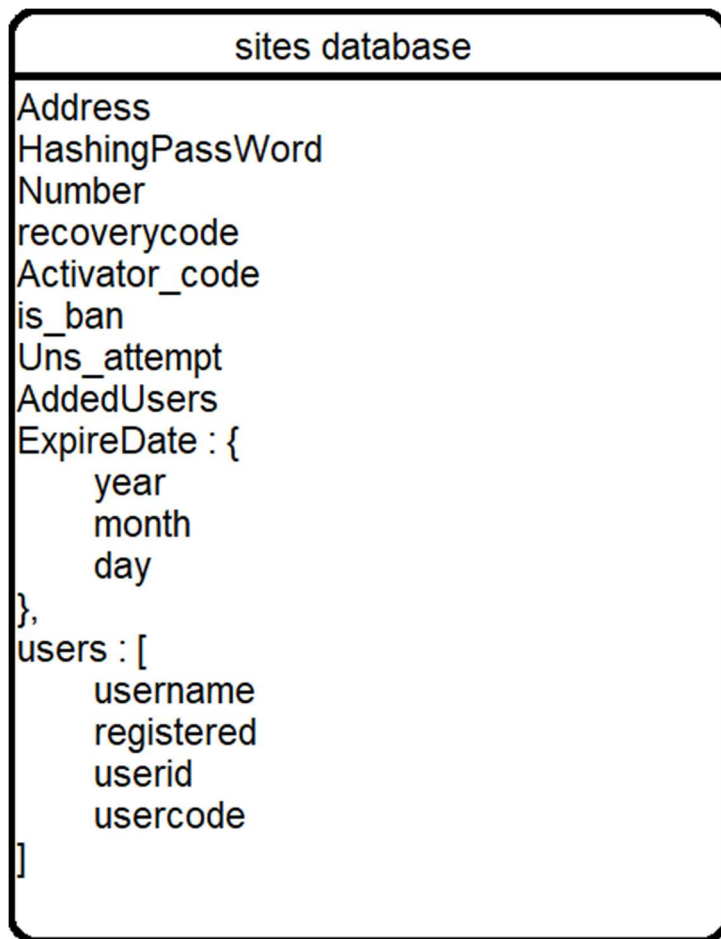
User Database

users database

```
fullName
Number
change_Number_code
Activator_code
Device_Id
change_Device_code
change_Device_Time
HashingPassword
Uns_attempt
is ban
recoverycode
EmailAddress
Email_Activator_code
Added_Email
Del_Acc_Link
sites:[
    SiteAddress
    username
    HashingCode
    expireTime
    is_expired
]
```

- fullName: user FullName
- Number: user phone Number
- Change_Number_code: a code that use for changing phone Number
- Activator_code: a code that use for Activation Account after registration or ban
- Device_Id: each Account need to be Active on special Device and cant and it is Device Id of that Device
- change_Device_code: a code that use for changing Device of Account
- change_Device_Time: each change_Device_code is active for 10 minute
- HashingPassword: The password is stored as a hash
- Uns_attempt: each unsuccesfull try to login, increases this variable by one unit
- is_ban: true is ban and false is not ban
- recoverycode: the code that use for recovery Account for Activation
- EmailAddress: what I say ? 😊
- Email_Activator_code: for Activation of Email use this code
- Added_Email: true if Email is Active and false if Email is not Active
- Del_Acc_Link: it's a code that use for Delete Account without login and just with Email
- SiteAddress: what I say too? 😊
- username: a username that use user at that site
- HashingCode: a code that use for login (main code)
- expireTime: each code has 1 minute time to use
- is_expired: is code is expire or not?

Sites Database



Fields without user databases fields:

- Address: site Address
- AddedUsers: each site can Add previous users one time
- year & month & day: expire Account time
- username: username of user at this site
- registered: true is This site Account Added to user Account
- userId: phone Number of user
- usercode: the code that use for Adding site to user Account

How to work

Signup and Activation Account

- 1-enter Full Name and Phone Number and password
- 2-click on link that sent to phone Number

Add site to user Account

- 1-signup to a site
- 2-get entirely code from that site (with QR code or manual)
- 3-enter code, username for that site and site Address (if enter with QR code, all of them is automatic)

Get and confirm code

- 1-Login to a site
- 2-Login to paas Application and get code from that site box
- 3-Write code in the password request section and send it

Delete Account with Email

- 1-enter Email
- 2-confirm it from clicking on the link that sent to Email
- 3-enter Email Address in the Email section on the site
- 4-click on link that sent to Email

Change Device

- 1-login to your account
- 2-click on "change device"
- 3-scan QR code from new Device

Change Number

- 1-login to your Account
- 2-click on "change Number"
- 3-input new Number
- 4-confirm Number with clicking on the link that sent to new Number

User Signup

Method: post

Address: localhost:4000/signup

Input fields:

- fullname
- Number
- DeviceId
- Password

Rules:

- The DeviceId is not received from the user and is read automatically from device

Responses:

- res.status(403).json({msg:'Unsuccessful'}) : Unsuccessful 😞
- res.status(403).json({msg:'this Number is already exist'}) : the Number is repetitive and is already exist
- res.status(403).json({msg:'Wrong Phone Number format'}) : the phone Number format is wrong(first Number must be 0 and its length must be 11 Numbers)
- res.status(403).json({ msg:'enter true fullname'}) : the name must not be empty
- res.status(403).json({msg:'Enter longer password'}) : password must be 8 characters or more
- res.status(200).json({msg:'user created'}) : successful

POST localhost:4000/signup

Params Authorization Headers (8)

● none ● form-data ● x-www-form-urlencoded

```
1 {
2   "fullname": "mohammad salehi",
3   "Number": "09385209351",
4   "password": "12345678",
5   "DeviceId": "A20mohammad"
6 }
```

ody Cookies Headers (8) Test Results

Pretty Raw Preview Visualize

```
1 {
2   "msg": "user created"
3 }
```


User login

Method: post

Address: localhost:4000/login

Input fields:

- Number
- DeviceId
- Password

Rules:

- The DeviceId is not received from the user and is read automatically from the device. And if it is different from the previous one, The server does not give tokens.

Responses:

- `res.status(403).json({msg:'Wrong_Device_ID'})` : DeviceId is different from previous DeviceId.
- `res.status(401).json({msg:'Auth Failed!'})` : something is wrong.
- `res.status(403).json({msg:'Unsuccessful'})` : Unsuccessful
- `res.status(403).json({msg:'ban'})` : Account is ban.
- `res.status(200).json({
 msg:'login_Successful',
 token:token
})` : login is Successful and Tokens given in the Tokens field.

POST localhost:4000/login

Send Save

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

raw JSON Beautif

```
1 {  
2   "Number": "09385209351",  
3   "password": "12345678",  
4   "DeviceId": "A20mohammad"  
5 }
```

Body 200 OK 375 ms 503 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "msg": "login_Successful!",  
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJ0dW1iZXIiOiIwOTM4NTIwOTM1MSIsImZ1bGx0YW11IjoibW9oYW1tYWQgc2FsZWhpI  
iwiaWF0IjoxNjEzMTYxMTU5LCJleHAiOjE2MTMxNjQ3NT19.  
V8UekNfDhkF0o3ajbBKzKF7_igXT3_LYMqjfxglwi0"  
4 }
```

Active User Account

Account need to be activated After signing up and become ban.

Method: get

Address: localhost:4000/Number(user Number)/code(sent to phone Number)

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Account is Active'})`

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, URL: localhost:4000/ActiveAccount/09385209351/pwt00cyhs5c12l6ctlar5eveo8mjsqm0befcc0u3bqal6pof4h918pv5pdd3
- Params Tab:** Labeled "Query Params", it contains a table with 3 columns: KEY, VALUE, and DESCRIPTION.

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Response Bar:** Status: 200 OK, Time: (blank)
- Response Body:** Displayed in JSON format:

```
1 {  
2   "msg": "Account is Active"  
3 }
```

Add Site to User Account

The user must first be added by the same site.

Method: post

Need the Token

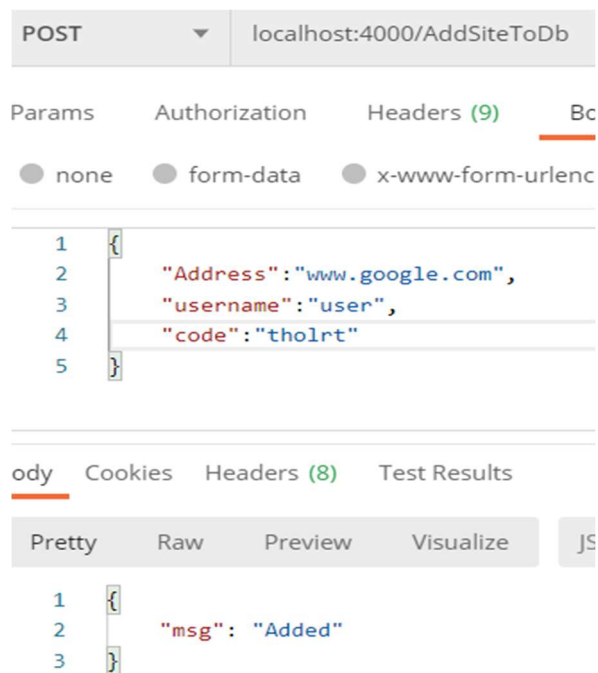
Address: localhost:4000/ AddSiteToDb

Input fields:

- Address
- username (User username on the same site)
- code (give from site and sent by QRcode or manual input)

responses:

- `res.status(404).json({msg:'need_add_to_db_from_site'})` : The site has not Added that user to DataBase
- `res.status(403).json({msg:'you are already added this site'})` : the site is already added
- `res.status(403).json({msg:'wrong code'})` : the code is wrong
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Added'})`



Get the password

Method: get

Need the Token

Address: localhost:4000/getcode

Input fields:

- Address (The site that the user wants to login)

Responses:

- `res.status(404).json({msg:'Wrong_Address'})` : site Address is wrong
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({
 msg:'Done',
 code:code
})`

The screenshot shows a REST client interface. At the top, the method is set to 'GET' and the URL is 'localhost:4000/getcode'. Below this, there are tabs for 'Params', 'Authorization', and 'Headers (9)'. Under 'Params', there are three radio buttons: 'none' (selected), 'form-data', and 'x-www-form-urlencoded'. The 'Body' tab is active, showing a JSON body with the following content:

```
1 {  
2   "Address": "www.google.com"  
3 }
```

Below the body tab, there are tabs for 'Cookies', 'Headers (8)', and 'Test Results'. The 'Test Results' tab is active, showing a 'Pretty' view of the response body:

```
1 {  
2   "msg": "Done",  
3   "code": "184307"  
4 }
```

Send recovery link

When Account is banned need to use it

Method: post

Address: localhost:4000/sendrecoverylink

Input fields:

- Number (user Number)

Responses:

- res.status(403).json({msg:'wrong Number'})
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({msg:'link sent'}) : link sent to phone Number

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL 'localhost:4000/sendrecoverylink' is entered. Below this, there are tabs for 'Params', 'Authorization', 'Headers (8)', and 'Body'. The 'Body' tab is selected, showing three radio buttons: 'none', 'form-data', and 'x-www-form-urlencoded'. The request body is a JSON object:

```
{  "Number": "09385209351"}
```

. Below the request, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected, showing response format options: 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The 'JSON' option is selected, displaying the response body:

```
{  "msg": "link sent"}
```

Get user Account Information

Method: get

Need the Token

Address: localhost:4000/getcode

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({
 msg:'Done',
 fullName:fullName,
 Number:Number,
 sites:postsites
})`

*** postsites is an Array that take a list of sites that user Added to his Account

The screenshot shows a REST client interface. At the top, a GET request is configured for the URL `localhost:4000/userinfo`. Below the URL bar, there are tabs for Params, Authorization, Headers (9), Body, and Pre-headers. The Body tab is selected, and the content type is set to 'raw'. The response body is displayed in a code editor with line numbers 1 through 3, showing a partial JSON object: `{` on line 1, on line 2, and `}` on line 3. Below the response body, there are tabs for Body, Cookies, Headers (8), and Test Results. The Body tab is selected, and the response is formatted as JSON. The response is displayed in a code editor with line numbers 1 through 11, showing a complete JSON object:

```
1 {  
2   "msg": "Done",  
3   "fullName": "mohammad salehi",  
4   "Number": "09385209351",  
5   "sites": [  
6     {  
7       "SiteAddress": "www.google.com",  
8       "username": "user"  
9     }  
10  ]  
11 }
```

Delete Account

Method: delete

Address: localhost:4000/DeleteAcc

need Token

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'User_Deleted'})`

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'DELETE' and the URL 'localhost:4000/DeleteAcc' is entered. Below this, there are tabs for 'Params', 'Authorization', and 'Headers (9)'. Under the 'Authorization' tab, three radio buttons are visible: 'none' (selected), 'form-data', and 'x-www-form-url'. The 'Headers' tab is active, showing a list of headers with line numbers 1, 2, and 3. Below the headers, there are tabs for 'body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'body' tab is selected, showing a JSON response in 'Pretty' format. The response is a JSON object with a single key 'msg' and the value 'User_Deleted'.

```
DELETE localhost:4000/DeleteAcc
```

Params Authorization Headers (9)

☒ none ☐ form-data ☐ x-www-form-url

```
1 {
2
3 }
```

body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize

```
1 {
2   "msg": "User_Deleted"
3 }
```

Add Email to Account (send link)

Method: post

Need the Token

Address: localhost:4000/AddEmail

Input fields:

- Email

Responses:

- `res.status(403).json({msg:'Wrong_Email'})` : Email format is wrong
- `res.status(200).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Email_Added'})`

The screenshot shows a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL is 'localhost:4000/AddEmail'. Below this, there are tabs for 'Params', 'Authorization', 'Headers (9)', and 'Body'. The 'Body' tab is selected and underlined in red. Under the 'Body' tab, there are three radio buttons: 'none', 'form-data', and 'x-www-form-urlencoded'. The 'x-www-form-urlencoded' option is selected. Below the radio buttons, a JSON object is displayed in a code editor with line numbers 1, 2, and 3. The JSON is:

```
{ 1: { 2:   "Email": "mohammad7979salehi@gmail.com" 3: }
```

Below the code editor, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected and underlined in red. Below the tabs, there are four buttons: 'Pretty', 'Raw', 'Preview', and 'Visualize'. The 'Pretty' button is selected. To the right of these buttons is a dropdown menu set to 'JSON'. Below the buttons, the JSON response is displayed in a code editor with line numbers 1, 2, and 3. The JSON is:

```
1: { 2:   "msg": "Email_Added" 3: }
```

Activator link for Email ➡ Inbox x

mohamadsalehi473@gmail.com

to me ▼

localhost:4000/ActiveEmail/09385209351/rv75yggmk7hmhy837w6wwfyq4v6h1vig8j7oo342plku62rtz2agymddw7zy

Active Email

Method: get

Address: localhost:4000/ActiveEmail/(user Number)/(user code)

Responses:

- `res.status(403).json({msg:'Wrong_link'})` : wrong code
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Email_is_Active'})`

GET

localhost:4000/ActiveEmail/09385209351/oliiy3re55ltypvsidmgembjctxhqj76u4e6b0ms2vwgzjtzdbbfmudzqg52

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

body

Cookies

Headers (8)

Test Results

Status: 200 OK

Time:

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "msg": "Email_is_Active"
3 }
```

Show QR Code (for change Device Id)

Method: get

Need the Token

Address: localhost:4000/showQR

Input fields:

- DeviceId

Responses:

- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({
 msg:'Done',
 Number:Number,
 code:code
}) (show this info as QRCode)

The screenshot displays a REST client interface. At the top, a GET request is configured for the URL `localhost:4000/showQR`. Below the URL bar, tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Se are visible. The 'Body' tab is selected, showing a JSON payload: `{ "DeviceId": "A21mohammad" }`. Below the body tab, radio buttons for content types (none, form-data, x-www-form-urlencoded, raw, binary, GraphQL) are shown, with 'raw' selected. The response section at the bottom has tabs for body, Cookies, Headers (8), and Test Results. The 'body' tab is active, showing a JSON response: `{ "msg": "Done", "Number": "09385209351", "code": "xcvmf72cuj2svyxzv901az0qwn8a5a975oit3ql91vf3b0fhi41ctsg9vh4p" }`. The response is formatted as JSON and includes a status icon.

```
GET localhost:4000/showQR
```

Params Authorization Headers (9) **Body** Pre-request Script Tests Se

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL

```
{  
  "DeviceId": "A21mohammad"  
}
```

body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "msg": "Done",  
3   "Number": "09385209351",  
4   "code": "xcvmf72cuj2svyxzv901az0qwn8a5a975oit3ql91vf3b0fhi41ctsg9vh4p"  
5 }
```

Get QRCode for change DeviceId

Method: post

Need the Token

Address: localhost:4000/getQR

Input fields: (automatically)

- Number (Account phone Number)
- code ()
- DevidId

Responses:

- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({msg:'changed'})

The screenshot displays a REST client interface with the following components:

- Request Section:**
 - Method: POST
 - URL: localhost:4000/getQR
 - Body tab selected, showing raw JSON:

```
{  "Number": "09385209351",  "code": "h07sw6du6rqi0nptlxxfvqmlwqo129q6nw2zs5cve7u241yes1jtd4v57g9q",  "DeviceId": "A21"}
```
- Response Section:**
 - Body tab selected, showing raw JSON:

```
{  "msg": "changed"}
```

Get link for change Number

Method: post

Need the Token

Address: localhost:4000/newNumber

Input fields:

- newNumber

responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'link sent'})`

The screenshot displays a REST client interface with the following details:

- Request:**
 - Method: POST
 - URL: localhost:4000/newNumber
 - Body: `{ "newNumber": "09166366715" }`
- Response:**
 - Status: 200 OK
 - Time: 306 ms
 - Size: 432 B
 - Body: `{ "msg": "link sent", "link": "localhost:4000/ChangeNumber/09385209351/09166366715/lv1yjtiz5pmd5un48gtl4jje4r0ken5lume3kkq4olfq4ict44051qduags", "Number": "09166366715" }`

Change Number with link

Method: get

Address: localhost:4000/(Account Number)/(new Number)/(code)

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Number changed'})`

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, Address: localhost:4000/ChangeNumber/09385209351/09166366715/lv1yjtiz5pmd5un48gtl4jje4r0ken5lumee3kkq4olfq4ict4405k...
- Request Tabs:** Params, Authorization, Headers (9), Body (selected), Pre-request Script, Tests, Settings.
- Request Body:** Content-Type: none, form-data, x-www-form-urlencoded, raw (selected), binary, GraphQL, JSON. The body content is an empty JSON object: `{}`.
- Response Bar:** Status: 200 OK, Time: 361 ms.
- Response Tabs:** Body (selected), Cookies, Headers (8), Test Results.
- Response Body:** Pretty, Raw, Preview, Visualize. The response is in JSON format: `{ "msg": "Number changed" }`.

Recovery password

Method: get

Address:localhost:4000/(mode (user or site))/(ID (user number or site Address))/(code)

Input fields:

- newPassword
- DeviceId

Responses:

- res.status(200).json({msg:'please choose longer password'})
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({msg:'changed'})

The screenshot displays a REST client interface. At the top, a GET request is configured for the URL `localhost:4000/recovery/users/09166366715/kb1qiuqd8ncl2uzo7subhb2a4qwhk6ts22yda2sae6m8osw6suiayk7y6hgz`. The 'Body' tab is selected, showing a JSON payload:

```
{  "newPassword": "11112222",  "DeviceId": "A21"}
```

. Below the request, the 'Test Results' tab shows a successful response with status `200 OK` and a time of `750`. The response body is displayed in 'Pretty' format as a JSON object:

```
{  "msg": "changed"}
```

Site Registration

Method: post

Address: localhost:4000/siteregistration

Input fields:

- Address
- Number
- Password

Responses:

- `res.status(403).json({msg:'Enter longer password'})`
- `res.status(403).json({msg:'Wrong Address'})`
- `res.status(403).json({msg:'Wrong Phone Number format'})`
- `res.status(403).json({msg:'this Number is already exist'})`
- `res.status(403).json({msg:'this Address is already exist'})`
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'user created'})` (need to active account with link)

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:4000/signupForSites
- Params:** none
- Authorization:** none
- Headers:** 8
- Body:**

```
{
  "Address": "www.google.com",
  "Number": "09385209351",
  "password": "1234567812345678"
}
```
- Response:**

```
{
  "msg": "user created"
}
```

Active site Account

Method: get

Address: localhost:4000/ActiveSite/(site Address)/(code)

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'Account is Active'})`

GET

localhost:4000/ActiveSite/www.google.com/jbxvg63f8f118wh15uuqg1do833rpvnn5cxppqzstyampzwb3xqw13pbqrl

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

Cookies

Headers (8)

Test Results

Status: 200 OK

Time: 30

Pretty

Raw

Preview

Visualize

JSON

1

{

2

"msg": "Account is Active"

3

}

Site login

Method: post

Address: localhost:4000/loginForSites

Input fields:

- Address
- password

Responses:

- res.status(401).json({msg:'Auth Failed!'})
- res.status(403).json({msg:'ban'})
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({
 msg:'login_Successful!',
 token:token
})

POST localhost:4000/loginForSites Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {  
2   "Address": "www.google.com",  
3   "password": "1234567812345678"  
4 }
```

ody Cookies Headers (8) Test Results 🌐 Status: 200 OK Time: 318 ms Size: 501

Pretty Raw Preview Visualize **JSON** 🔗

```
1 {  
2   "msg": "login_Successful!",  
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJkZGRyZXNzIjoid3d3Lmdvb2dsZS5jb20iLCJ0dW1iZXIiOiIwOTM4NTIwOTM1MSIsIm1hdCI6MTYxMzI0OTU0NywiZXhwIjoxNjQ0ODAzMTQ3fQ.  
    DRvgUQ8c1tWVokq53VGGUswTGMEU0IdPW6pLOI-y4BY"  
4 }
```

Add users to site database

Method: post

Need the Token

Address: localhost:4000/AddUserToSiteDb

Input fields:

- username (user username for that site)

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(403).json({
 username:username,
 msg:'already_Added_from_before'
})`
- `res.status(200).json({
 username:username,
 msg:'user_Added',
 code:code
})`

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL 'localhost:4000/AddUserToSiteDb' is entered. Below this, there are tabs for 'Params', 'Authorization', 'Headers (9)', and 'Body'. The 'Body' tab is selected, showing a JSON payload: `{ "username": "example_user_1" }`. Below the body tab, there are radio buttons for 'none', 'form-data', and 'x-www-form-urlencoded'. Further down, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected, showing a JSON response: `{ "username": "example_user_1", "msg": "user_Added", "code": "ciqb6q" }`. The response is formatted in a 'Pretty' view.

```
POST localhost:4000/AddUserToSiteDb
```

Params Authorization Headers (9) **Body**

☒ none ☐ form-data ☐ x-www-form-urlencoded

```
1 {  
2   "username": "example_user_1"  
3 }
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "username": "example_user_1",  
3   "msg": "user_Added",  
4   "code": "ciqb6q"  
5 }
```

Confirm code

Method: get

Need the Token

Address: localhost:4000/confirm

Input fields:

- username
- code

Responses:

- res.status(404).json({msg:'wrong username'})
- res.status(404).json({msg:'wrong Address'})
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({
 msg:'login auth',
 username:username,
 authentication:false,
 value:'wrong code'
})
- res.status(200).json({
 msg:'login auth',
 username:username,
 authentication:false,
 value:'expire time error'
})
- res.status(200).json({
 msg:'login auth',
 username:username,
 authentication:false,
 value:'has been used before'
})
- res.status(200).json({
 msg:'login auth',
 username:username,
 authentication:true,
 value:'auth completed'
})

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:4000/confirm
- Params:** none, form-data, x-www-form-urlencoded (selected)
- Headers (9):** (tab selected)
- Request Body:**

```
1 {  
2   "username": "example_user_1",  
3   "code": "880159"  
4 }
```
- Response:** (tab selected)
 - Headers (8):** (tab selected)
 - Test Results:** (tab selected)
 - Response Body:**

```
1 {  
2   "msg": "login auth",  
3   "username": "example_user_1",  
4   "authentication": true,  
5   "value": "auth completed"  
6 }
```

Send recovery link for sites

Method: post

Address: localhost:4000/sendrecoverylinkforsites

Input fields:

- Address

Responses:

- `res.status(403).json({msg:'wrong Address'})`
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'link sent' })`

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL 'localhost:4000/sendrecoverylinkforsites' is entered. Below this, there are tabs for 'Params', 'Authorization', 'Headers (8)', and 'Body'. The 'Body' tab is selected, and within it, there are radio buttons for 'none', 'form-data', 'x-www-form-urlencoded', and a selected 'JSON' option. The JSON body is shown as:

```
1 {  
2   "Address": "www.google.com"  
3 }
```

Below the body tab, there are tabs for 'body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'body' tab is selected, and it shows a 'Pretty' view of the JSON response:

```
1 {  
2   "msg": "link sent"  
3 }
```

Sites Account renewal

Address: localhost:4000/renewal

Method: post

Need the Token

Input methods:

- mode (1= 1 month, 2= 3 month, 3= 6 month, 4= 1 year)

Responses:

- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({
 msg:'extentioned',
 year,
 month,
 day
})

The screenshot shows a REST client interface. At the top, the method is set to POST and the URL is localhost:4000/renewal. Below this, there are tabs for Params, Authorization, and Headers (9). Under Params, there are three radio buttons: none, form-data, and x-www-form-u. The request body is shown in a JSON editor with the following content:

```
1 {  
2   "mode": 4  
3 }
```

Below the request body, there are tabs for body, Cookies, Headers (8), and Test Results. The 'body' tab is selected, and it shows a JSON response in a Pretty view:

```
1 {  
2   "msg": "extentioned",  
3   "year": 1400,  
4   "month": 12,  
5   "day": 29  
6 }
```

Add previous users

Method: post

Need the Token

Address: localhost:4000/Addprevioususers

Input fields:

- users(Array of usernames)

Responses:

- res.status(403).json({msg:'already Added'}) : that site already added users
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({msg:'Added'})

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL 'localhost:4000/Addprevioususers' is entered. Below this, there are tabs for 'Params', 'Authorization', 'Headers (9)', and 'Body'. The 'Body' tab is selected, showing a JSON object:

```
{ 1 { 2   "users": ["user1", "user2", "user3"] 3 }
```

. Below the request, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected, showing a JSON response:

```
1 { 2   "msg": "Added" 3 }
```

Dashboard informations

Address: localhost:4000/dashboard

Method: get

Need the Token

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({
 msg:'Done',
 Address,
 Number,
 ExpireDate,
 users (Array)
})`

GET localhost:4000/dashbord

Params Authorization Headers (9) **Body**

☐ none ☐ form-data ☐ x-www-form-urlencoded

Body

ody Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2    "msg": "Done",
3    "Address": "www.google.com",
4    "Number": "09385209351",
5    "ExpireDate": {
6      "year": 1400,
7      "month": 12,
8      "day": 29
9    },
10   "users": [
11     {
12       "username": "example_user_1",
13       "registered": true
14     },
15     {
16       "username": "user1",
17       "registered": false
18     },
19     {
```

Get user code

Method: get

Need the Token

Address: localhost:4000/getusercode

Input fields:

- username

Responses:

- `res.status(404).json({msg:'user_not_founded'})`
- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(400).json({msg:'already_added'})`
- `res.status(200).json({
 msg:'not_Added',
 username,
 code
})`

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:4000/getusercode
- Params:** none
- Authorization:** none
- Headers:** 9 (not expanded)
- Body:**

```
{  
  "username": "example_user_2"  
}
```
- Response:**

```
{  
  "msg": "not_Added",  
  "username": "example_user_2",  
  "code": "yup5y8"  
}
```


Send delete Account link to Email

Method: post

Address: localhost:4000/sendDeleteLink

Input fields:

- Email

Responses:

- res.status(403).json({msg:'Email is not Active'})
- res.status(403).json({msg:'Unsuccessful'})
- res.status(200).json({msg:'link sent'})

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL 'localhost:4000/sendDeleteLink' is entered. Below this, there are tabs for 'Params', 'Authorization', 'Headers (8)', and 'Body'. The 'Body' tab is selected, showing a JSON object:

```
{  "Email": "mohammad7979salehi@gmail.c"}
```

. Below the request, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected, showing the response in 'Pretty' format:

```
{  "msg": "link sent"}
```

Delete Account by link

Method: get

Address: localhost:4000/DelByLink/(Email)/(code)

Responses:

- `res.status(403).json({msg:'Unsuccessful'})`
- `res.status(200).json({msg:'User_Deleted'})`

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method: GET, Address: localhost:4000/DelByLink/mohammad7979salehi@gmail.com/ib7dcjrpsqusokuq1scphb3vkio76fjeilqdfdw8ck2yjjg7o5...
- Params Tab:** Contains a table for Query Params.
- Body Tab:** Active tab showing the response body in JSON format.
- Status Bar:** Shows Status: 200 OK and Time: 355 ms.

Query Params Table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

Response Body (JSON):

```
1 {  
2   "msg": "User_Deleted"  
3 }
```