Service Layer Design Document

Service Layer is an abstraction over domain logic or over application's business logic. It defines application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation.

Use the Service Layer pattern when we need:

- To encapsulate domain logic under API.
- To implement multiple interfaces with common logic and data.

MVP Service end points:

1. Host/api/login
2. Host/api/register
3. Host/api/dashboard/home
4. Host/api/dashboard/manage_bills
5. Host/api/dashboard/Billing_events
6. Host/api/dashboard/reports
7. Host/api/dashboard/findings
8. Host/api/dashboard/manage_bills/add_bill
9. Host/api/dashboard/manage_bills/remove_bill
10. Host/api/dashboard/manage_bills/edit_bill
11. Host/api/dashboard/manage_bills/status_bill/1
12. Host/api/dashboard/manage_bills/status_bill/0
13. Host/api/ dashboard/reports/date
14. Host/api/ dashboard/findings/date
15. Host/api/ dashboard/due_date_alert

Description of end points:

1. Host/api/login - Used to make a post request with username and password arguments in json format with provide the authorization to User.

2. Host/api/register - used to sign up the user and store the user details.
3. Host/api/dashboard/home - used to fetch all the data related to home page of dashboard.
4. Host/api/dashboard/manage_bills - used to view manage bills page.
5. Host/api/dashboard/Billing_events - used to view the billing events page.

6. Host/api/dashboard/reports - used to view the reports page.
7. Host/api/dashboard/findings - used to view the findings page.
8. Host/api/ dashboard/manage_bills/add_bill - used to add billing details.
9. Host/api/ dashboard/manage_bills/remove_bill - used to remove billing details.
10. Host/api/ dashboard/manage_bills/edit_bill - used to edit billing details.
11. Host/api/ dashboard/manage_bills/status_bill/1 - used to change the billing status as active.
12. Host/api/ dashboard/manage_bills/status_bill/0 - used to change the billing status as inactive.
13. Host/api/ dashboard/reports/date-to pick reports for particular date.
14. Host/api/ dashboard/findings/date-to pick findings for any date.
15. Host/api/ dashboard/due_date_alert- to show notifications on the dashboard.

Sample request/response

1. Host/api/login:

Request(post):

{

“username”:”email/username”

“password”:”password”

}

Response:

{

Result:0/1

}

2. Host/api/register

Request(post)

{

“username”:value,

“email”:value,

“phone”:value,

“name”:value,

“password”:value,

“confirm-password”:value

}

Response

{

“result”:0/1

}

3. Host/api/dashboard/home

Request (get):no input parameters

Response:

{

Pending_bills:{

Card:[{

Bill_id:value,

Bill_amount:value

},

{

Bill_id:value,

Bill_amount:value

},

{

Bill_id:value,

Bill_amount:value

},

]

Bar_graph:{

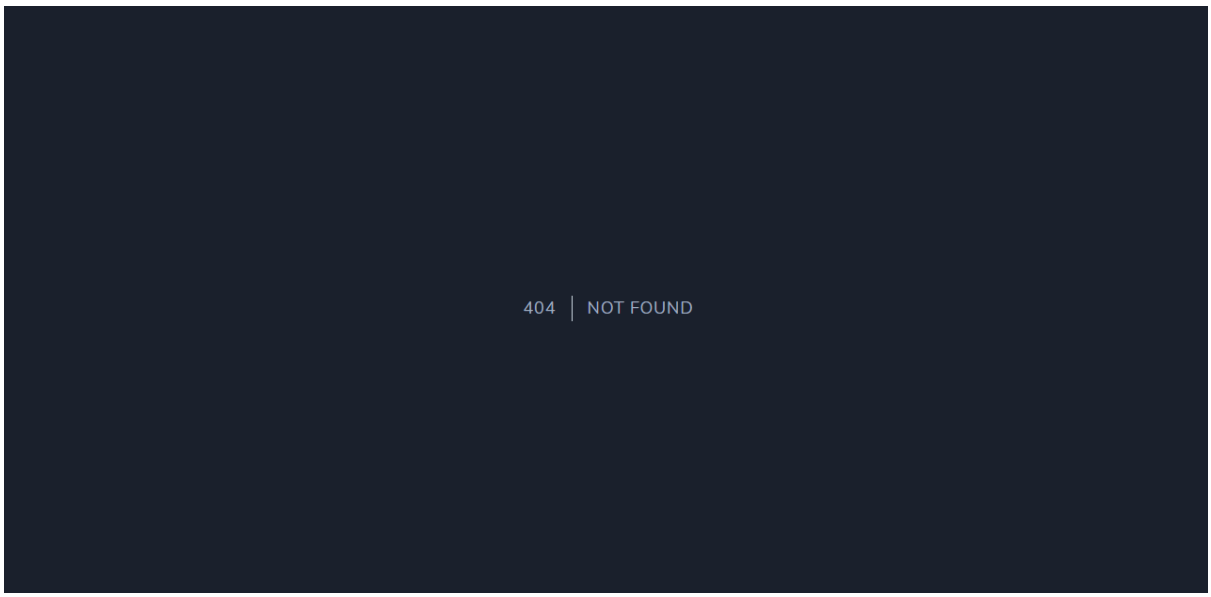
Sample

}

}

Response code:

1. 404: The 404 error status code indicates that the REST API can't map the client's URI to a resource but may be available in the future. Subsequent requests by the client are permissible



2. 401 unauthorised.
3. A 401 error response indicates that the client tried to operate on a protected resource without providing the proper authorization.
4. Internal server error: 500 is the generic REST API error response. Most web frameworks automatically respond with this response status code whenever they execute some request handler code that raises an exception.
5. 4.400 bad request- it is the generic client-side error status, used when no other 4xx error code is appropriate. Errors can be like malformed request syntax, invalid request message parameters, or deceptive request routing etc.
6. 5.200 ok: It indicates that the REST API successfully carried out whatever action the client requested and that no more specific code in the 2xx series is appropriate.

Diagrams for mapping endpoint to user interface: