API Service Setup

What is this API instance Request Intercepter Response Intercepter and is there any other Terms also present give me the Name

Great question, Prasun! Let's break it down simply — you're using **Axios**, and terms like `API Instance`, `Request Interceptor`, and `Response Interceptor` are part of **Axios' advanced features** to handle HTTP requests efficiently.

✅ 1. What is an API Instance (Axios Instance)?

It's like **creating a custom HTTP client** that has your preferred settings (base URL, headers, timeout, etc.) already set — so you don't have to repeat them every time.

Example:

```js
const axiosInstance = axios.create({
  baseURL: "http://localhost:8000",
  timeout: 10000,
  headers: {
    "Content-Type": "application/json",
  },
});
```

Now you can just use `axiosInstance.get(...)` or `axiosInstance.post(...)` without setting baseURL every time.

✅ 2. Request Interceptor

It **runs before the actual request is sent**. You can modify headers, add tokens, logs, etc.

Use Case:

Add `Bearer Token` for auth.

```js
axiosInstance.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem("token");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);
```

✅ 3. Response Interceptor

It **runs after you get a response** but before it reaches your `.then()` or `.catch()`.

Use Case:

Redirect to login if token is invalid or show global error message.

```js
axiosInstance.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);
```

✅ 4. Other Key Terms You Should Know:

Term	Description
Base URL	The default URL for every API call.
Timeout	Maximum time (ms) to wait for a response.
Headers	Extra info (like `Content-Type`, `Authorization`) sent with every request.
Middleware	Functions that sit in the middle to process data (more common in backend like Express, but similar idea).
Error Handling	Custom behavior when requests fail (like showing alerts or logging out).
Status Code Check	Handling based on response code (e.g., 200 OK, 401 Unauthorized, 500 Server Error).
Token Injection	Adding access tokens automatically in headers using interceptors.
Retry Logic	Automatically retrying failed requests (advanced use case).

If you want, I can draw a **diagram or flow** that shows how all these Axios concepts work together in a real project. Want that too?

I can draw a diagram or flow that shows how all these Axios concepts work together in a real project. yaml

Absolutely, Prasun! Here's a **YAML-based flowchart** that visually (textually) shows how all the **Axios concepts** — like instance, request interceptor, response interceptor, and API call — work together in a real-world project:

```yaml
Axios Workflow in a Real Project:

START: User Action (e.g., button click to fetch data)
  |
  v
[Frontend Component (React/Vue/Angular)]
  |
  v
[Axios API Call Triggered using axiosInstance]
  |
  v
[Axios Instance]
  - baseURL: http://localhost:8000
  - headers:
      Content-Type: application/json
  - timeout: 10000
  |
  v
[Request Interceptor]
  - Check localStorage for token
  - If token exists:
      Add "Authorization: Bearer <token>" to headers
  - Proceed with request config
  |
  v
[HTTP Request Sent to Backend]
  |
  v
[Backend Processes Request]
  - Matches route
  - Validates token
  - Fetches data from DB
  - Sends response
  |
  v
[Response Interceptor]
  - If status === 401:
      Redirect to /login
  - If status === 500:
      Show global error
  - Else: proceed with response
  |
  v
[Frontend Component Receives Data]
  - Update state/UI with response
  |
  v
END: User sees updated UI or error message
```

🔁 Optional Add-ons You Can Implement Later:

```yaml
Optional:
  Retry Logic:
    - On network error, retry 3 times before failing
  Logging:
    - Log every request and response globally
  Caching:
    - Cache GET responses for performance
```

Let me know if you want the same flow in image format too — I can generate that!

in more details

Absolutely Prasun! Here's a **more detailed and expanded YAML-style flow diagram** of how **Axios works inside a full-stack web app**, with all important parts included — **frontend triggering the request, Axios instance setup, request/response interceptors, backend interaction**, and what happens at each stage.

```yaml
Axios Full Workflow in a Web Application:

START: User Action
  description: User clicks a button or interacts with UI (e.g., Submit Login form)
  triggers: API call from frontend
  |
  v

[Component/Service Layer]
  description: Frontend function initiates API call using Axios instance
  example: axiosInstance.post(API_PATHS.AUTH.LOGIN, userData)
  |
  v

```

```
[Axios Instance]
  purpose: Centralized Axios config for reuse across app
  config:
    baseURL: http://localhost:8000
    timeout: 10000
    headers:
      Content-Type: application/json
      Accept: application/json
  |
  v

[Request Interceptor]
  purpose: Preprocess all outgoing requests
  tasks:
    - Read access token from localStorage
    - If token exists:
        - Add Authorization: Bearer <token> to request headers
    - Log request details (optional)
  result: Modified request config
  |
  v

[HTTP Request Sent]
  type: GET | POST | PUT | DELETE
  to: Full URL = baseURL + endpoint (e.g., /api/v1/auth/login)
  body:
    - Data from frontend (e.g., login form values)
  headers:
    - With added token if available
  |
  v

[Backend Server Receives Request]
  stack: Express.js / Django / Spring Boot / etc.
  steps:
    - Match the route and HTTP method
    - Run middleware (e.g., authentication check)
    - Parse request body and validate input
    - Query/Update the database (MongoDB, SQL, etc.)
    - Prepare response data (JSON format)
  |
  v

[HTTP Response Sent Back]
  structure:
    - status: 200 | 401 | 404 | 500, etc.
    - data: JSON payload (user info, error message, etc.)
  |
  v

[Response Interceptor]
  purpose: Process responses before it reaches component
  tasks:
    - Check for status codes (401, 500, etc.)
    - If 401:
        - Redirect user to /login
    - If 500:
        - Show global toast or console error
    - Otherwise:
        - Pass response data to next step
  |
  v

[Frontend Component Receives Response]
  success:
    - Extracts data
    - Updates UI (e.g., show dashboard, render chart)
  error:
    - Catches errors in .catch()
    - Shows alert/toast or logs error
  |
  v

END: User sees the result of the action
  successExample: "Welcome, Prasun!" message or data table rendered
  errorExample: "Invalid credentials" shown via toast or modal
```

🧠 Bonus Section: Common Advanced Features

yaml

```
Advanced Features (Optional to Add in Axios):
  Retry Logic:
    - Automatically retry failed requests with delay
    - Use axios-retry or write custom retry handler

  Global Loading Spinner:
    - Trigger "loading=true" on request
    - Set "loading=false" on response/error

  Token Refresh Logic:
    - If token expired (403):
        - Try refreshing token using refreshToken API
        - Retry original request

  Error Logger:
    - Log all API errors to external logging services (e.g., Sentry)

  Caching:
    - Cache GET requests to avoid redundant API calls
```

Would you like me to convert this into a visual flowchart image next? That could make this whole process super clear to visualize!