Java Backend Development Live-85

lecture-19

Agenda

- Minor-project-2 (cont.)
  - Redis in Minor-project
  - Improve existing APIs
- OAuth2
  - Intro
  - Application Registration
  - Client ID and Client Secret
  - Example: GitHub, Google, LinkedIn
- Monolithic vs Microservices architecture
- Consume Rest API (RestTemplate)

What is OAuth 2.0?

OAuth2 is an open **authorization protocol**, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as Facebook, GitHub,GFG, etc. It allows sharing of resources stored on one site to another site without using their credentials.

It works by **delegating user authentication** to the service that hosts a user account and **authorizing third-party applications** to access that user account.

OAuth2 Roles

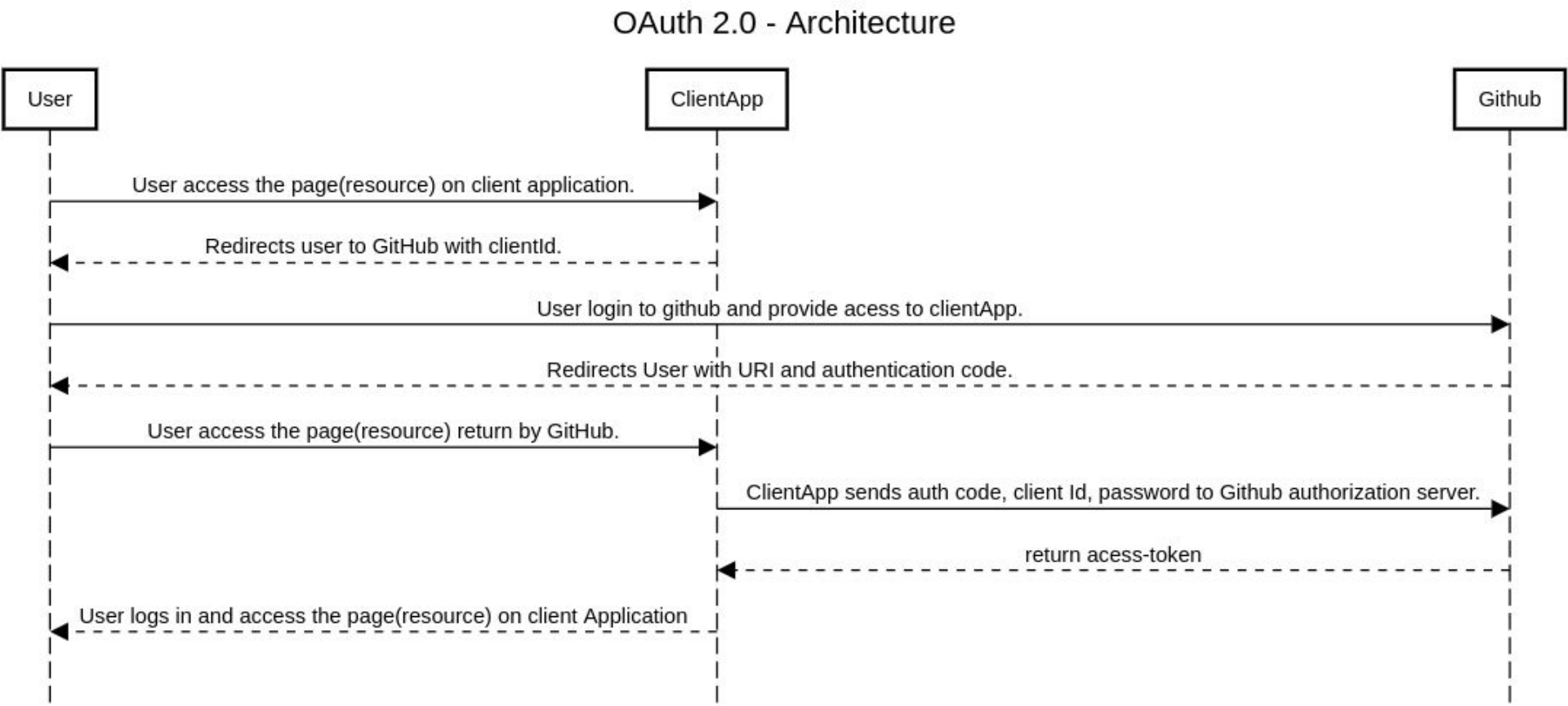
Resource Owner: The resource owner is **the user** who authorizes an application to access their account. The application's access to the user's account is limited to the scope of the authorization granted (e.g. read or write access)

Client: The client is the application that wants to access the user's account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.

Resource Server: The resource server hosts the protected user accounts.

Authorization Server: The authorization server verifies the identity of the user then issues access tokens to the application.

OAuth 2.0 - Architecture



Application Registration

Before using OAuth2 with your application, you must register your application with the service.The following information is needed for registration.

- Application Name
- Application Website(home page)
- Redirect URI or Callback URL

Client ID and Client Secret

The **Client ID** is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users.

The **Client Secret** is used to authenticate the identity of the application to the service API when the application requests to access a user’s account, and must be kept private between the application(geeksforgeeks/JBDL-85) and the API(github).

Examples

Github: https://github.com/settings/developers

LinkedIn: https://developer.linkedin.com/

Facebook: https://developers.facebook.com/products/facebook-login/

Google: https://console.cloud.google.com/apis/credentials/

Project Setup

References:

https://spring.io/guides/tutorials/spring-boot-oauth2/

https://github.com/settings/developers

https://datatracker.ietf.org/doc/html/rfc6749#section-3.3

Record Last Access Time

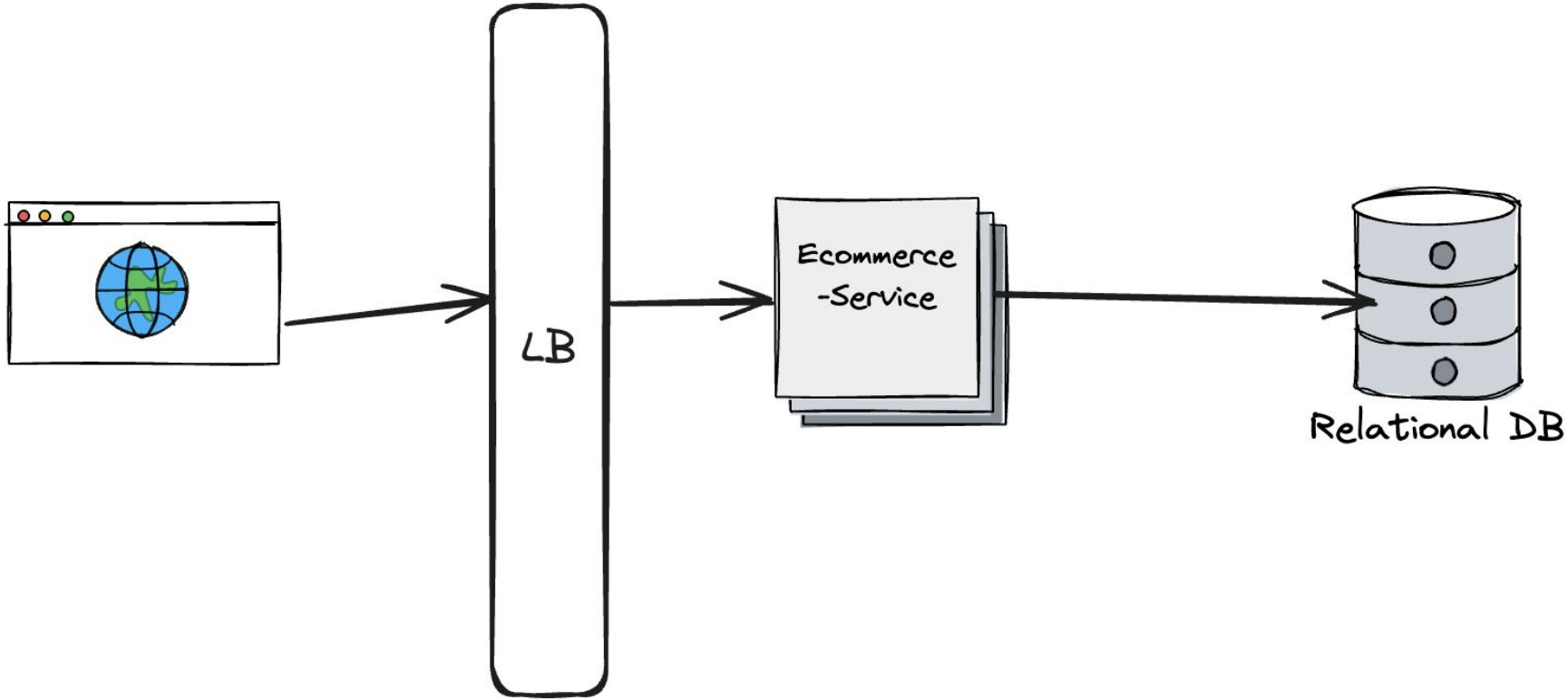
Create a HttpFilter.

Read Oauth2User from SecurityContextHolder.

Fetch user from database.

Update lastAccess field in the database.

Monolithic architecture



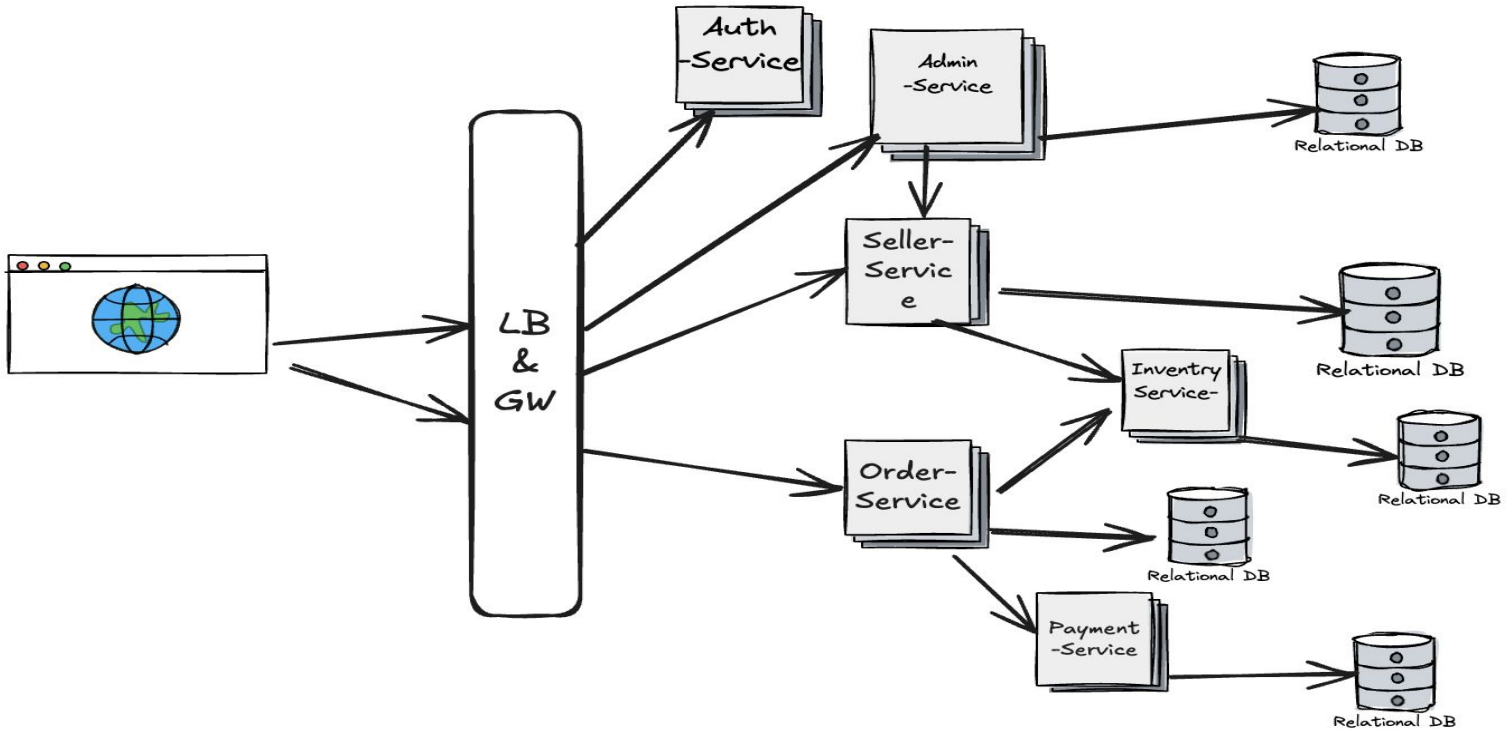
Pros of monolithic applications:

- Simpler to develop and deploy
- Easier to test
- Requires less specialized skills
- Singular security management

Cons of monolithic applications:

- Can become complex over time
- Difficult to scale
- Technology limitations
- Single point of failure
- CI-CD will take more time.

Microservices based architecture



Pros of microservices application:

- Self-contained services:Each microservice is self-contained, meaning it can be debugged, deployed, and managed independently of other modules.
- Easy to scale
- More flexibility:Teams can more easily add additional functionality and new technologies to a microservices-based architecture as needed.

Cons of microservices applications:

- Increased complexity
- Requires specialized skills
- Distributed security and testing
- Extra costs

Resource: https://martinfowler.com/articles/microservices.html

Consume Rest API (Rest Template)

The **Rest Template** is the central Spring class used to create applications that consume RESTful Web Services.

You can use the methods available in the Rest Template class to consume the web services for all HTTP methods.

Sample API: https://jsonplaceholder.typicode.com/posts/23