**Nginx as Load Balancer:**

* Nginx serves as the main load balancer, forwarding requests to instances of the API Gateway.
* API Gateway instances are dynamically updated through service discovery (Consul), eliminating the need for manual IP address entries.

**API Gateway:**

* Instances of the API Gateway forward requests to microservices based on the request URL.
* No validation is performed at the API Gateway; it acts solely as a request forwarder.
* Service discovery (Consul) automatically determines routes, reducing manual configuration.

**Microservices Communication:**

* Microservices communicate by directly sending requests from one service to another, bypassing the API Gateway. This approach ensures a loosely coupled architecture, allowing each microservice to interact independently.
* Service discovery (Consul) plays a crucial role in facilitating dynamic routing. Microservices use Consul to discover the locations of other services, eliminating the need for static configurations. This dynamic road determination enhances flexibility and scalability within the system.

**User Authentication with Keycloak:**

* Keycloak is employed for user registration, authentication, and token issuance.
* Supports various authentication methods, including credentials (login/password) and social media logins (e.g., Google).

**Email Verification Process:**

* To enhance the account activation process, user email verification is implemented.
* Due to the time-consuming nature of Keycloak's email sending, an external "Email" microservice is utilized.
* RabbitMQ is employed as a message queue to streamline and expedite the email verification process.
* Microservices needing to send emails leverage the centralized "Email" microservice for efficiency and consistency.

**Security Measures:**

* After successful authentication, Keycloak issues two tokens:
* An access token (JWT) with a short expiration time for enhanced security.
* A refresh token with a relatively long expiration time.
* When the access token expires, the refresh token is utilized to request the issuance of a new access token.
* This token refresh process is seamlessly handled by Keycloak, ensuring continuous and secure access to the system without requiring the user to re-authenticate.
* Each microservice (resource server) independently validates access tokens (JWT) issued by the Keycloak server **locally**. This ensures that token validation occurs in place, directly on the resource server, eliminating the need for sending tokens to the Keycloak server for validation. Decisions regarding request permission are made based on information extracted from the access token.

**User Interface Handling:**

* HTML, CSS, and JS files are stored directly on each microservice, ensuring a decentralized structure.
* Each microservice contains only the templates required for its specific functionality, minimizing unnecessary dependencies.
* Currently, there is no dedicated UI server; the user interface components are distributed across the microservices.
* This approach facilitates a modular and self-contained design, allowing each microservice to manage its own user interface components efficiently.

**Note:**

Try to deploy multiple instances of Keycloak in containerized environments. When authentication is required, the system tries to forward the request to one of the available Keycloak instances, ensuring reliability and load balancing for authentication processes.  
  
Make separate Admin Api gateway which will forward admin requests  
  
  
  
The way to organize docker configuration:  
  
so I will have configuration files on my local machine(Windows).   
  
In my docker-compose.yml I specify the dockerfile which will be used for configuring this image.  
  
In my dockerfile I will specify the operation I need ( such as update,installs etc.)   
Also I specify the mount directories (connect my local files with directories on docker container)