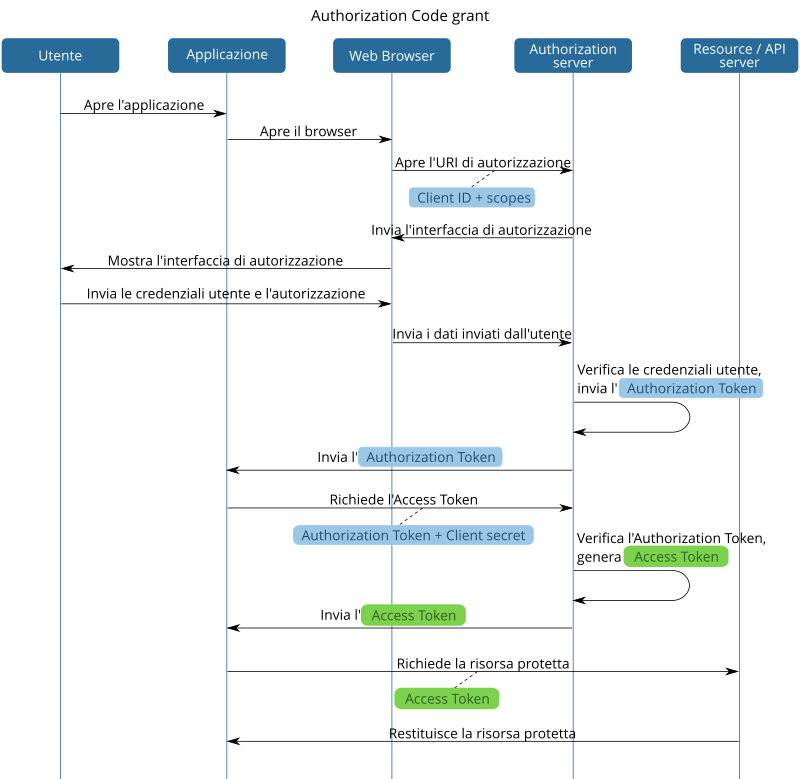
# **REPORT CYBERSECURITY**

L’obbiettivo è quello di creare un server rest API con dati finti e casuali da affiancare ad un servizio reale e critico che nel nostro caso riguarda il sistema informatico di un ospedale.  
Al giorno d’oggi molte applicazioni sono considerabili come un front-end per una serie di chiamate API. Le API sono necessarie per il corretto funzionamento di un’applicazione e se non sono protette adeguatamente attaccanti malevoli potrebbero esfiltrare dati, effettuare attacchi DDoS ai server o abusarne.  
Per proteggere le nostre API abbiamo utilizzato il protocollo **Oauth** che permette l’accesso alle risorse ad utenti autenticati. In particolare abbiamo utilizzato l’authorization code flow che viene rappresentato nella seguente immagine.



Per rendere ulteriormente più sicuro il protocollo abbiamo utilizzato l’estensione PKCE, il quale consiste nella generazione di un code verifier ed un code challenge che permettono di evitare un’eventuale intercettazione da parte di un attaccante dell’authorization code.

**Authorization Server**

Per gestire il processo di autenticazione l’authorization server dispone di tre endpoint:

* /auth
* /signin
* /token

**/auth** : il client invia il proprio client\_id, redirect\_url e code\_challenge per avviare il processo di creazione di un nuovo authorization code e conseguente access token. In particolare l’authorization server andrà a verificare se il client\_id e redirect\_url corrispondono ai dati salvati nella precedente fase di registrazione, se corretti si viene reindirizzati a /signin.  
[CODICE]

**/signin** :viene presentata all’utente una pagina di login per la verifica delle sue credenziali personali. Se la verifica ha successo viene creato un **authorization** **code** in cui viene memorizzato client\_id, redirect\_url, code\_challenge, username e expiration\_date. Al termine viene eseguito un redirect a redirect\_url inviando l’authorization code.

[CODICE]

**/token** : il client invia il proprio client\_id, client\_secret, redirect\_url, code\_verifier e authorization code all’authorization server, il quale verifica la validità dell’authorization code e sfrutta il code verifier per controllare se il code\_challenge inviato precedentemente è corretto. In caso di successo viene restituito un access token, con il quale l’applicazione per conto dell’utente può accedere a determinate risorse del server API.

[CODICE]

L’authorization sever offre anche una pagina dedicata alla registrazione di nuove applicazioni che vogliono utilizzare le API. Effettuata la registrazione viene restituito un client\_id e un client\_secret

**Server API**

Per la realizzazione del server API abbiamo utilizzato fastAPI, un framework moderno e ad alte prestazioni per la creazione di API.  
Abbiamo definito diverse routes raggiungibili solamente se l’utente autenticato possiede i diritti per accedervi. Per questo motivo abbiamo inserito diversi ruoli che permettono di simulare un sistema reale in cui a tipi di utenti differenti corrispondono privilegi differenti. Nel nostro caso specifico abbiamo tre ruoli: dottore, infermiere, paziente (ciascuno di essi appartiene ad uno specifico reparto).  
In questo modo ogni volta che il client invia una richiesta al server API si verifica se il token di accesso è ancora valido e in caso affermativo viene fatta una richiesta all’authorization server per ricavare lo username dell’utente a cui è associato l’access token. Lo username viene quindi utilizzato per ricavare il ruolo e il reparto di appartenenza dell’utente e stabilire se ha il permesso per accedere alla risorsa richiesta.  
Per la generazione di dati casuali abbiamo utilizzato llama come LLM, il quale si occuperà di fornire i dati riguardanti dottori, infermieri e pazienti.

-aggiungere immagine iniziale con architettura sistema (client, authServer, APIServer)  
-aggiungere dettagli sul flow (PKCE, redirect\_url, session)  
-aggiungere parti di codice  
-aggiungere descrizione API  
-spiegare funzionamento client registration.