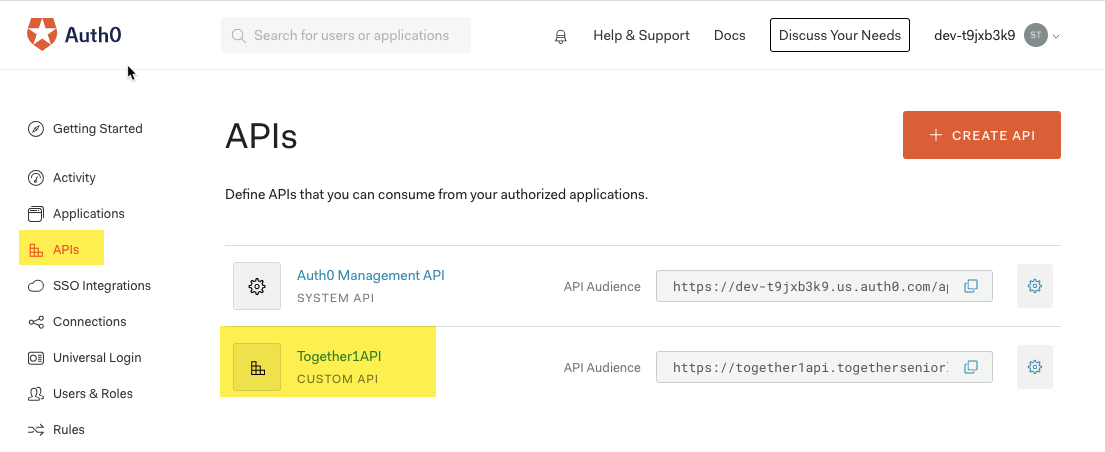
### Overview of RBAC in Together1

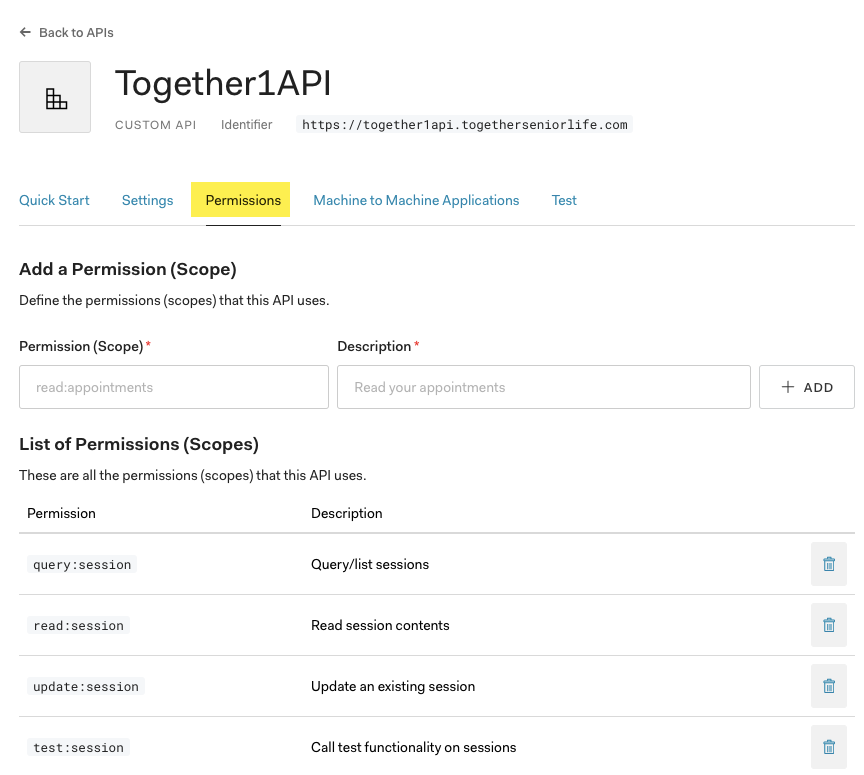
We are implementing this based on the RBAC features of Auth0. All users will have an identity in Auth0, which will host the login dialog so users can login via username/password. For some users (for example, likely the case of those participating in the research classes) we can create logins for them, and use the “Passwordless Connections” feature of Auth0 to automagically log them in, without password, simply by following a link in an email or SMS.

In Auth0, we can define our own set of specific, discrete permissions for an API. For example, we have one server-side API which is for Angular to call within the video platform.

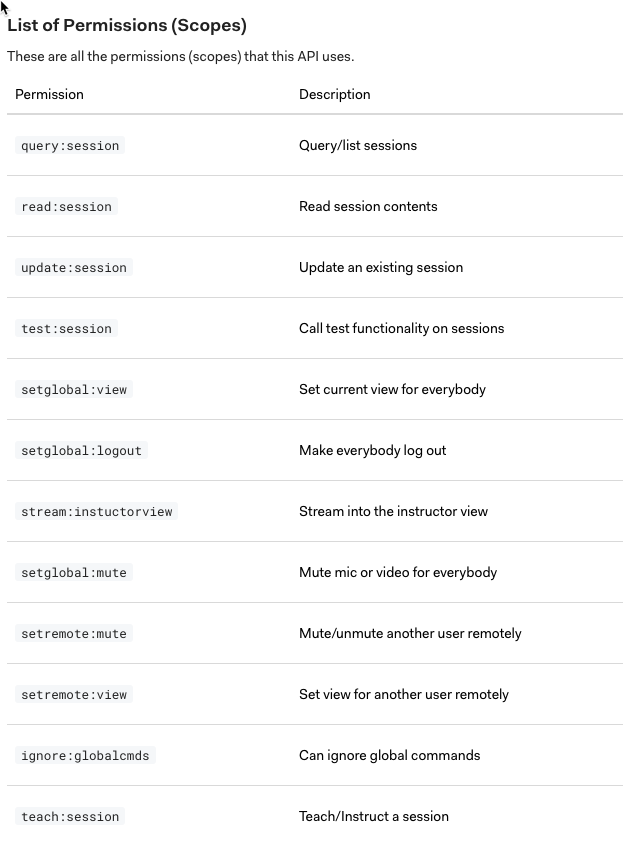
Permissions are specific to an API in Auth0. To find the permissions, first navigate to “APIs/Together1API”:



Then check the “Permissions” tab:



Examples of permissions that we have defined for this API include:



Roles are defined by a set of permissions; users belonging to a role inherit the permissions defined by the permissions of the role. Users can belong to multiple roles, thus inheriting the superset of all permissions assigned to their roles.

Permissions can allow us to have finer grained control over what users can see and do; we should almost always create logic around permissions, not roles. For example, the ‘setglobal:view’ permission gives the user the ability to set the current video session view (group vs. instructor) for all participants in a video session. From a business perspective, we may state that instructors have the ability to set the session view for the particular session, and provide a button, visible only to instructors, to switch the view. However, the underlying implementation does not actually check the current role of the user to set the visibility of the button, but instead checks that the current user has the corresponding permissions required to invoke the operation(s) invoked by the button.

Enabling features and enforcing access based on permissions instead of roles gives us the flexibility to implement RBAC based on simpler, permission-based logic and the flexibility to modify access simply by reassigning roles and the permissions assigned to roles with minimal code changes.

### 

### Accessing User Permissions In Angular

The process of authenticating the user within the browser passes a JWT access back to the browser, which is made available by the Auth0 Single-Page Application SDK. This *Bearer token* is signed by Auth0, and so the authenticity and integrity of the token’s data is assured.

Here is an example of the JSON data contained within a token (decoded with ‘jsonwebtoken’ package on the server):

| JWT Token: {  "header": {  "alg": "RS256",  "typ": "JWT",  "kid": "IMPsSxChys4t-kfV4gsa-"  },  "payload": {  "iss": "https://dev-t9jxb3k9.us.auth0.com/",  "sub": "auth0|5f15a837965b8c0019fe6367",  "aud": [  "https://together1api.togetherseniorlife.com",  "https://dev-t9jxb3k9.us.auth0.com/userinfo"  ],  "iat": 1595793891,  "exp": 1595880291,  "azp": "NBDvM2hcVzkvwbtnuNNT3qQCe71kIbt2",  "scope": "openid profile email",  "permissions": [  "query:session",  "read:session",  "setglobal:logout",  "setglobal:mute",  "setglobal:view",  "setremote:mute",  "setremote:view",  "stream:instuctorview",  "test:session",  "update:session"  ]  },  "signature": "R2mPjcXPCm5fdQSu0vgYy4BBbUvn3Qr4gil1vnMNXfe9CKjFJ6bG6yRtn0UaT8Gzo6GwzEv6ByGUSeWtWKTK59B0G-T9yjyiAW8SnHy\_oJWxKPHHsDAmqXAbxKlX8fQqxb2VykYNNzx3dn6j\_MqRzs6VUKyyqbeKFIWkC56W7QeLwutz8PrmrD\_IiYXGHYQkUeeHY2YWIgQt8CUCUrbVIcxUdKgy9hMg8BEVXmZc7sGKwwtDPneIiAoensMS8rVz4UPoAbzn72d3C15mF9yQpt8dilNiEzFWrOsmJAyHJvq3e2Typw\_abi1EP1XNedcJEIUv\_zALc\_DmuzoVgtpGVQ" } |
| --- |

Notice the permissions property, which contains a list of permissions granted to the user. In this example, the list includes “setglobal:view”, which indicates that the user has been granted permission to set the session view for all participants.

### 

### Securing Access to the Service-Side API

In Together1, the Bearer token is retained and passed to the server-side API via an HTTP injector that adds the token to the ‘Authenticated’ HTTP header. On the server, the npm package ‘express-jwt’ is an Express middleware component that validates the token before passing the request onto controllers. In addition, the token can be decoded and the permissions contained therein used to enforce authorization for the API.

### Notes on Token Validation

### Resources

[Auth0 Node.js SDK](https://auth0.github.io/node-auth0/)

[Auth0 SDK for Single Page Applications](https://auth0.github.io/auth0-spa-js/)

[Tokens](https://auth0.com/docs/tokens)

[Navigating RS256 and JWKS](https://auth0.com/blog/navigating-rs256-and-jwks/)

[auth0/node-jwks-rsa: A library to retrieve RSA public keys from a JWKS (JSON Web Key Set) endpoint.](https://github.com/auth0/node-jwks-rsa)

[Jsonwebtoken](https://www.npmjs.com/package/jsonwebtoken)

[Brute Forcing HS256 is Possible: The Importance of Using Strong Keys in Signing JWTs](https://auth0.com/blog/brute-forcing-hs256-is-possible-the-importance-of-using-strong-keys-to-sign-jwts/)