DevAuth: Device Based Federated Login

Project Team: Ravinder Singh, Prateek Narendra, Kai-Chieh Huang, Nishant Shankar

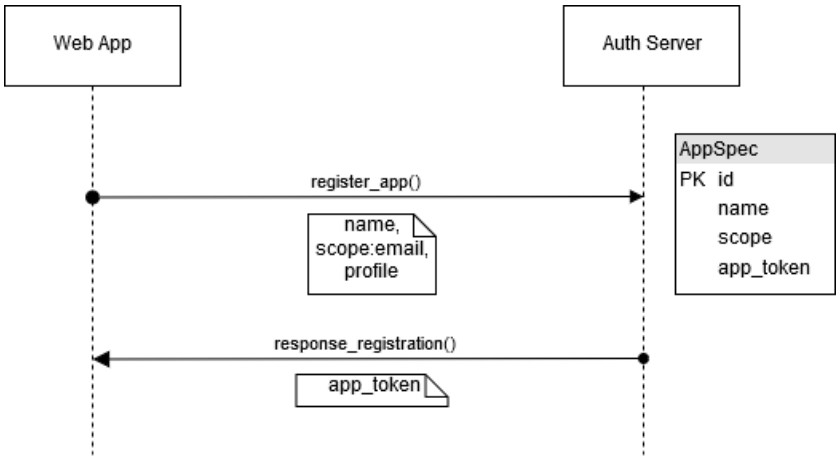
Progress

By referring to the OpenID spec (as discussed in the literature review), we zeroed in on the design of our system. There are three players in our system apart from the end-user: a client application, an authorization server and the user's mobile application. We formalized the different flows that allow for device-based authentication, considering comparable ease of use to OpenID. These flows include:

1. Client App Registration
2. Device Registration
3. Signup/Login of User on Web App

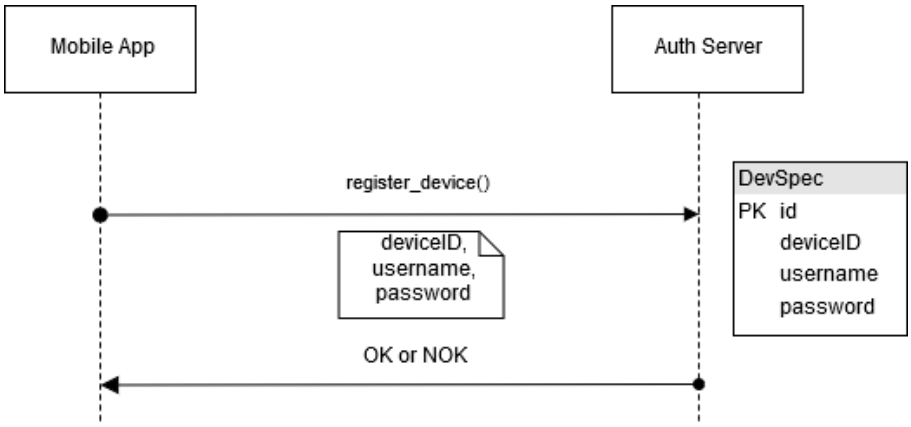
Client App Registration

Client Apps in our system are applications that wish to utilize DevAuth as a means of user authentication [1]. They can register with the Auth server by specifying a certain scope, and a unique client ID is returned. The Auth server keeps a mapping of client apps and IDs. Communication happens over HTTPS, ensuring security is maintained. So far, we have built a client application that sends requests to the Auth Server for registration. In our Django codebase, the AppManagement project deals with client app registration/retrieving ID tokens.



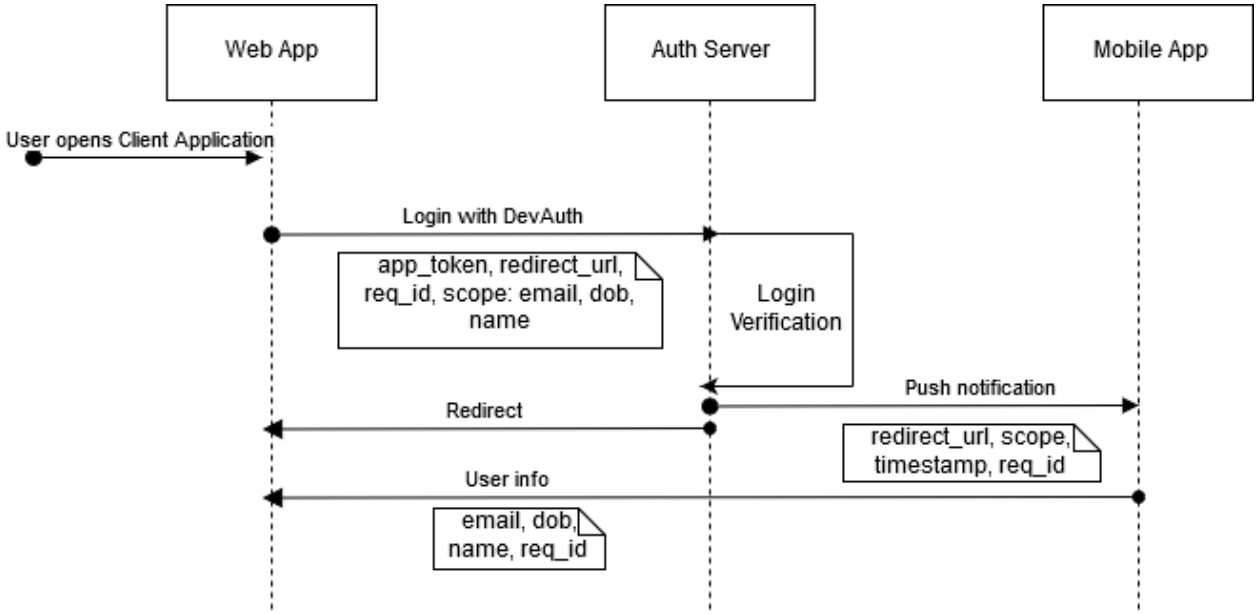
Device Registration

The device registration flows are similar to the client app registration. The initiator, in this case, is a mobile application. We employed React Native to create an Android Application. The user can register their device with DevAuth by using this app and can add relevant information like username and password.



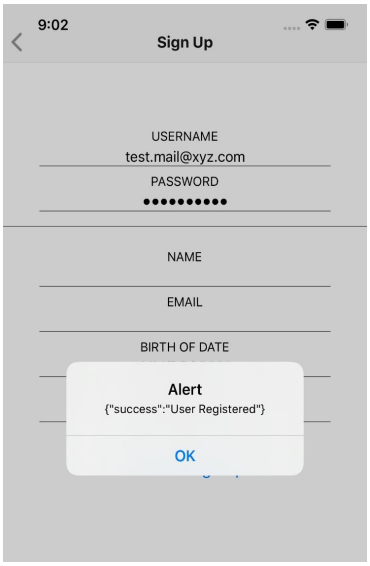
Signup/Login of User on Web App

The flow closely relates to the OpenID/OAuth flow except for the authorization server being the the mobile device.

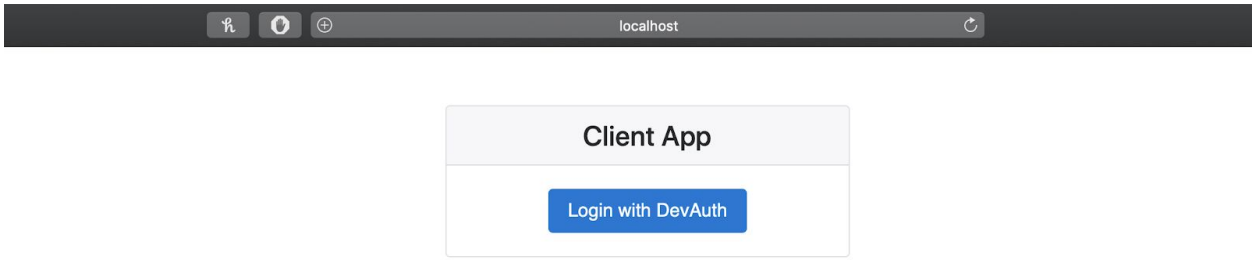


Snapshots/Preliminary Results

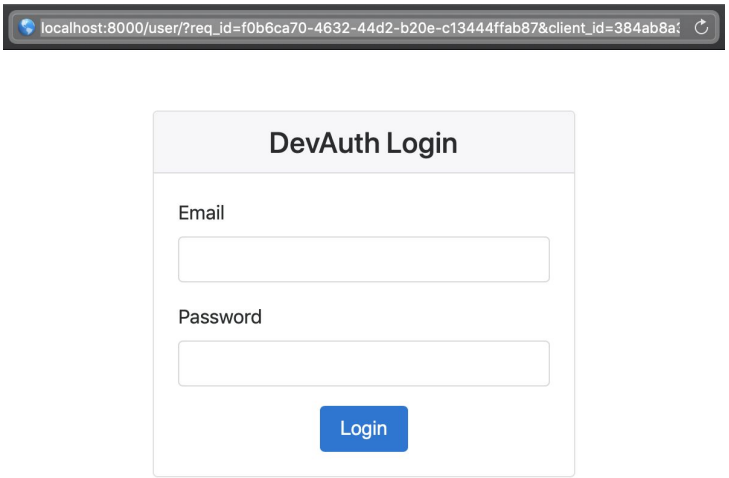
Registration of the User on Mobile Phone



Sample Client Application



Redirection After Clicking Login



Tests Performed:

1. Client registration with the AuthServer.
2. Redirection from Client App to the Login page of the AuthServer.
3. User Registration with the Auth Server, through
4. Login of the User on the AuthServer.

Link to Postman suite, used to test the application :
https://github.com/Ethos-lab/research-project-f19-http-404/blob/master/code/APIRequests/AuthServerRequests.md.md

Schedule

By mid-November, we will be able to finish the implementation of our established flow. The tasks that remain are,

1. Pushing notification to the Mobile app from the AuthServer.
2. Sending the response from the mobile app to the Client App.
3. Redirection from the AuthServer to the Client App in response received from the Mobile app.

Task	Expected Deadline
Pushing notification	11-12-2019
Response to Client App	11-15-2019
Redirection	11-20-2019

Obstacles

Due to our design, we feel that we can completely eliminate passwords from the flow, as we only require access to a phone to authenticate. In this design, we are moving from using “Something that we know” to “Something that we have”. This can be disadvantageous in certain scenarios (when user doesn’t have access to their device). Moreover, heavy reliance on mobile devices leads to some accessibility issues, so the mobile app will have to be designed with that in mind.

Session Management - The major issues we need to look into is Token Refresh Compliance specified by [2].

During development, we chose to develop our Client Mobile Application for iOS platform due to prior knowledge and experience. However, iOS Application development has significant initial costs.

Workarounds

We were initially using public keys and some encryption schemes to encrypt our data. But now we are using HTTPS to secure the communication between the client app and the Auth server. We decided to port our Native application to Android to save development costs.

References:

1. [RFC - 6749] - https://tools.ietf.org/html/rfc6749
2. [RFC - 6454] - https://www.rfc-editor.org/rfc/rfc6454.txt