FREE eBook

LEARNING odata

Free unaffiliated eBook created from
Stack Overflow contributors.

#odata

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: odata

It is an unofficial and free odata ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official odata.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with odata

Remarks

This section provides an overview of what odata is, and why a developer might want to use it.

It should also mention any large subjects within odata, and link out to the related topics. Since the Documentation for odata is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting odata set up or installed.

Odata- The Best way to Rest

From Odata.org

> OData (Open Data Protocol) is an OASIS standard that defines the best practice for building and consuming RESTful APIs. OData helps you focus on your business logic while building RESTful APIs without having to worry about the approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats and query options etc. OData also guides you about tracking changes, defining functions/actions for reusable procedures and sending asynchronous/batch requests etc. Additionally, OData provides facility for extension to fulfil any custom needs of your RESTful APIs.

The most exciting feature of Odata from my view are,

1. **Url conventions**

In normal API's there is no standard way to specify a url, means by seeing someone API we cannot ensure that what that API is doing. Odata helps to create a standard url based on the business logic.

2. **Querying**

In normal API, once we created further if we want only a specific data from the response we will do like this

- Call the API (From Server API returns everything)
- Apply filtering on the client side.

otherwise creates separate API which results filtered data.

But instead of these OData API allows querying option means we can include filtering conditions in

Odata url itself, the Odata automatically filter the result from server so we can acheive what ever data we wants alone.

To play with Odata in postman http://www.odata.org/getting-started/learning-odata-on-postman/

Read Getting started with odata online: https://riptutorial.com/odata/topic/5830/getting-started-with-odata

Chapter 2: Azure AD authentication for Node.js

Introduction

This document provides a high level overview and explains the whole architecture of Azure AD Authentication Process for Node.js (MOBILE TOOL) It is explains technical component and its interaction between mobile App, Web-API, Document DB and Azure Active Directory. Finally, how mobile user will be able to login into system and perform operations. The document provides a high-level description of the goals of the architecture, the use cases support by the system and architectural styles and comp

Examples

Content

High Level Design Document

MOBILE Tool (Azure AD authentication for Node.js)

Version 1.0 (Draft)

Table of Contents

High Level Design (HLD)

1. Introduction This document provides a high level overview and explains the whole architecture of Azure AD Authentication Process for Node.js (MOBILE TOOL) It is explains technical component and its interaction between mobile App, Web-API, Document DB and Azure Active Directory. Finally, how mobile user will be able to login into system and perform operations. The document provides a high-level description of the goals of the architecture, the use cases support by the system and architectural styles and components that have been selected to best achieve the use cases. 1.1. Purpose High Level Design Document (HLD) provides a comprehensive architectural overview of MOBILE APP. 1.2. Scope The scope of this HLD is to depict the architecture of the MOBILE APP developed by Preludesys team.

1.3. Definitions, Acronyms, and Abbreviations

- HTTP – Hypertext Transfer Protocol
- Azure AD – Azure Active Directory
- Web-API – Web Application Program Interface
- REST API – Representational state transfer
- JSON – JavaScript Object Notation
- ADAL - Azure Active Directory Authentication Library
- iOS – Operating system for Apple Devices
- Android - Operating system for Google Devices
- OAuth2 /OpenID- an open standard for authorization
- User - This is organization user having member of Azure Active Directory.
- Creator (Process Owner) - This is a user who can create/ modify Document
- Reader - This user can read/view Data
- Administrator – this user can read, modify and delete any of DTCPII tool Process Specification, Process Review, Process Template and administer other user rights / roles. Administrator can delegate or share administrative rights to other users in the system.

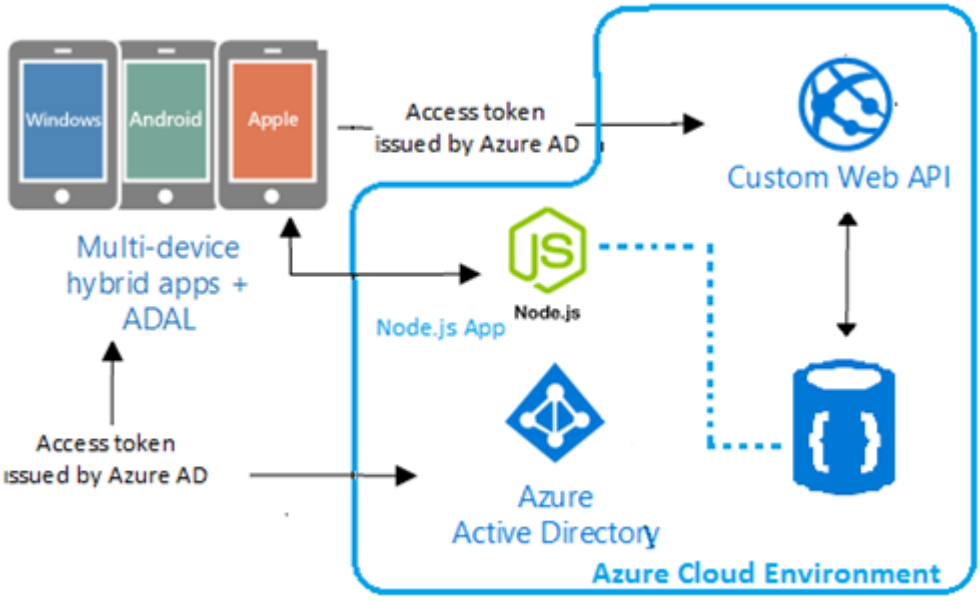
1.4. References

1.5. Overview

This document provides technical overview of following components.

- Node.JS Module
- Custom Web API
- ADAL Library
- Azure Active Directory
- Document DB

2. Architectural Representation (Logical View)

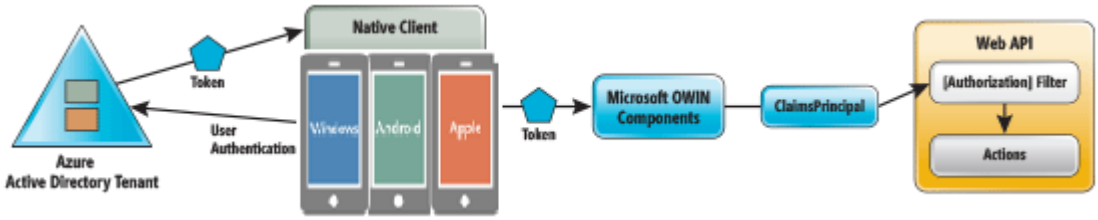


The solution is divided into two components.one is Client side app which would run on mobile and Server side component running in Azure environment.

Mobile App (Node.js Client) Node.js is light weight responsive user interface. A mobile application would simply be the, written in the native language or using a hybrid solution like Cordova (aka PhoneGap). This user interface would get data and perform business logic via calls to a server side API. If mobile app requires portions of the business logic, it'd have to be rewritten in objective-C for iOS and Java for Android. While Cordova offers a solution for some code reuse, it cannot leverage Node or its ecosystem directly.

Server Side Component Node.js Server Node is a JavaScript web server runtime. This act as Web server for mobile app. It use JSON as data exchange format, JSON, can be natively parsed by both ends. Node differs from most web application runtimes in the manner that it handles concurrency. Rather than using threading to accomplish concurrency, Node relies on an event-driven loop running in a single process. Node supports an asynchronous (non-blocking) model. Web API This is REST API solution running on Azure as Web API. Its running on top of Document DB and act as protective layer for Document DB. ASP.NET tooling and security middleware from the Microsoft Open Web Interface for .NET (OWIN) components that make protecting Web API straightforward. The end point is protected using azure AD OAuth Bearer 2.0 Access Tokens.

3. Technical Process



3.1. Authentication Authentication boils down to asking a caller, when sending a message to a server, to include some kind of credential that can verify his identity or retrieve his attributes. The server then uses that information for authorization—determining whether access should be granted and in what terms. Resources often offload most of the authentication functions to an external services provider, commonly known as an authority or an identity provider. These providers take care of heavy-duty tasks such as onboarding users, assigning credentials, handling lifecycle flows (such as password recovery), providing a UI for user authentication, credential verification over multiple protocols, multiple authentication factor management, fraud detection and more. With those functions out of the way, the only authentication task left is to verify that authentication succeeded at the authority of choice. This typically involves examining a security token, a data fragment issued by an authority to a caller upon successful authentication. Security tokens are usually crafted according to a specific format, digitally signed by a key that will unequivocally identify the issuing authority, and contain some data that uniquely ties the token to the target resource. When the resource receives a request, it looks for an accompanying token. If it finds one that meets the required validation attributes, the caller is authenticated. At that level of detail, the pattern is so generic it can describe many different approaches to authentication. I'll apply it to this scenario by assigning the high-level roles to concrete entities. The Resource The resource will be the ASP.NET Web API 2 project I need to secure. You can apply authentication requirements with finer granularity.

3.2. Custom Web API and secure End point The Authority will configure the Web API to offload its authentication needs to Windows Azure AD, a Platform as a Service (PaaS) available to every Windows Azure subscriber. Windows Azure AD is specifically designed to support cloud-based

application workloads. The service stores information about users (attributes and credentials) and organizational structure. It can synchronize data with Windows Server Active Directory, provided connect exclusively in the cloud. 3.3. OAuth Token Format OAuth 2.0 Validation process doesn't mandate any particular token format, but the JSON Web Token (JWT) format (bit.ly/14EhlE8) for REST scenarios has become the de facto standard. Both Windows Azure AD and the Microsoft OWIN components support the JWT format in OAuth 2.0 flows. The mechanics of JWT acquisition and validation are all taken care of by the middleware, and the token format is transparent to the application code. 3.4. Node.js Module with ADAL Library Node.js modules is a mobile interface contains UI components and authentication module to get security token from Active Directory using ADAL library. The ADAL for node.js library makes it easy for node.js applications to authenticate to Azure AD in order to access Azure AD protected web resources.

3.5. Document DB Setup and Configuration

3.6. Azure AD sync using Azure AD Connect This is out of scope as business users are already part of office 365 which implies they already sync with Azure Active Directory.

4. Architectural Goals and Constraints Server side Mobile tool will be hosted on Azure Cloud platform running as Web Application All communication with client has to comply with public HTTPS, TCP/IP communication protocol standards. Client Side Users will be able to access Mobile tool over Android and iOS devices.

4.1. Security

To be able to authenticate users, you must register Node.js application with the Azure Active Directory (AAD). This is done in two steps. First, you must register your mobile service and expose permissions on it. Second, you must register your iOS app and grant it access to those permissions 4.2. Persistence

Data persistence will be addressed using Azure Document DB database.

4.3. Reliability/Availability

4.4. Performance

There is no particular constrains related to system performance. It is anticipated that the system should respond to any request well under standard database and web server script timeouts (20 seconds), also system performance can depend on available hardware, PSU network and internet connection capabilities. In addition, upload / download times can depend on data size which in turn depends on user input. Therefore, actual performance can be determined only after system deployment and testing.

5. Size and Performance

6. Issues and concerns

Read Azure AD authentication for Node.js online: https://riptutorial.com/odata/topic/10583/azure-ad-authentication-for-node-js

Credits

S. No	Chapters	Contributors
1	Getting started with odata	Community, LMK
2	Azure AD authentication for Node.js	Sandip Dalvi