MS-600 Outline Notes

Sunday, August 9, 2020 3:25 PM

Exam MS-600: Building Applications and Solutions with Microsoft 365 Core Services – Skills Measured

NOTE: The bullets that appear below each of the skills measured are intended to illustrate how we are assessing that skill. This list is not definitive or exhaustive.

NOTE: In most cases, exams do NOT cover preview features, and some features will only be added to an exam when they are GA (General Availability).

Implement Microsoft Identity (20-25%)

Register an Application

- determine the supported account type
- select authentication and client credentials for app type and authentication flow
- define app roles

Implement Authentication

- configure Microsoft Authentication Library (MSAL JS) for endpoint and token cache
- plan and configure scopes for dynamic or static permission
- use the MSAL JS login method

Configure Permissions to Consume an API

- configure Delegated permissions for the app
- configure Application permissions for the app
- identify admin consent requirements

Implement Authorization to Consume an API

- configure incremental consent scopes
- call MSAL JS using AquireTokenSilent/AquireToken pattern

Implement Authorization in an API

- validate Access Token
- configure effective permissions for delegated scopes
- implement app permissions using roles
- use a delegated access token to call a Microsoft API

Create a Service to Access Microsoft Graph

Account types in the public cloud

In the Microsoft Azure public cloud, most types of apps can sign in users with any audience:

- If you're writing a line-of-business (LOB) application, you can sign in users in your own organization. Such an application is sometimes called *single-tenant*.
- If you're an ISV, you can write an application that signs in users:
  - In any organization. Such an application is called a *multitenant* web application. You'll sometimes read that it signs in users with their work or school accounts.
  - With their work or school or personal Microsoft accounts.
  - With only personal Microsoft accounts.
- If you're writing a business-to-consumer application, you can also sign in users with their social identities, by using Azure Active Directory B2C (Azure AD B2C).

Account type support in authentication flows

Some account types can't be used with certain authentication flows. For instance, in desktop, UWP, or daemon applications:

- Daemon applications can be used only with Azure AD organizations. It doesn't make sense to try to use daemon applications to manipulate Microsoft personal accounts. The admin consent will never be granted.
- You can use the Integrated Windows Authentication flow only with work or school accounts (in your organization or any organization). Integrated Windows Authentication works with domain accounts, it and requires the machines to be domain joined or Azure AD joined. This flow doesn't make sense for personal Microsoft accounts.
- The Resource Owner Password Credentials grant (username/password) can't be used with personal Microsoft accounts. Personal Microsoft accounts require that the user consents to accessing personal resources at each sign-in session. That's why this behavior isn't compatible with non-interactive flows.

Account types in national clouds

Apps can also sign in users in national clouds. However, Microsoft personal accounts aren't supported in these clouds. That's why the supported account types are reduced, for these clouds, to your organization (single tenant) or any organizations (multitenant applications).

From <https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-supported-account-types>

https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-acquire-cache-tokens

MSAL caches a token after it's been acquired. Your application code should first try to get a token silently from the cache before attempting to acquire a token by other means.

MSAL allows you to get tokens to access Azure AD for developers (v1.0) and Microsoft identity platform (v2.0) APIs. v2.0 protocol uses scopes instead of resource in the requests.

When your application needs to request an access token with specific permissions for a resource API, pass the scopes containing the app ID URI of the API in the format <app ID URI>/<scope>.
Some example scope values for different resources:

- Microsoft Graph API: https://graph.microsoft.com/User.Read
- Custom web API: api://11111111-1111-1111-1111-111111111111/api.read

The format of the scope value varies depending on the resource (the API) receiving the access token and the aud claim values it accepts.
For Microsoft Graph only, the user.read scope maps to https://graph.microsoft.com/User.Read, and both scope formats can be used interchangeably.

Declare app roles using Azure portal

1. Sign in to the Azure portal.
2. Select the **Directory + Subscription** icon in the portal toolbar.
3. In the **Favorites** or **All Directories** list, choose the Active Directory tenant where you wish to register your application.
4. In the Azure portal, search for and select **Azure Active Directory**.
5. In the **Azure Active Directory** pane, select **App registrations** to view a list of all your applications.
6. Select the application you want to define app roles in. Then select **Manifest**.
7. Edit the app manifest by locating the appRoles setting and adding all your Application Roles.

A client application gains access to a resource server by declaring permission requests. Two types are available:

- "Delegated" permissions, which specify scope-based access using delegated authorization from the signed-in resource owner, are presented to the resource at run-time as "scp" claims in the client's access token.
- "Application" permissions, which specify role-based access using the client application's credentials/identity, are presented to the resource at run-time as "roles" claims in the client's access token.

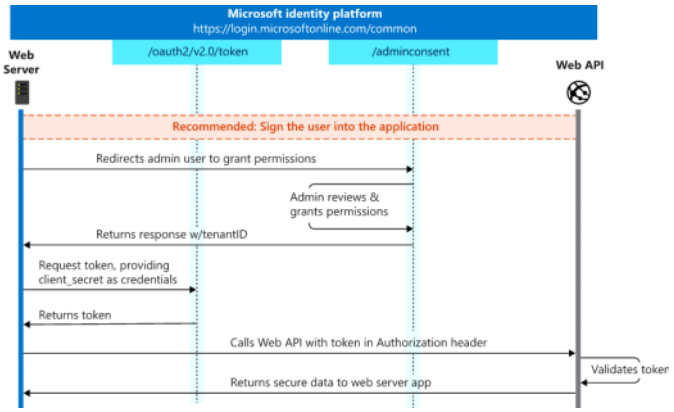
```
function authCallback(error, response) {
    // Handle redirect response
}

userAgentApplication.handleRedirectCallback(authCallback);

const accessTokenRequest: AuthenticationParameters = {
    scopes: ["user.read"]
}

userAgentApplication.acquireTokenSilent(accessTokenRequest).then(function(accessTokenResponse) {
    // Acquire token silent success
    // Call API with token
    let accessToken = accessTokenResponse.accessToken;
}).catch(function (error) {
    //Acquire token silent failure, and send an interactive request
    console.log(error);
    if (error.errorMessage.indexOf("interaction_required") !== -1) {
        userAgentApplication.acquireTokenRedirect(accessTokenRequest);
    }
});
```

A JWT contains three segments, which are separated by the . character. The first segment is known as the **header**, the second as the **body**, and the third as the **signature**. The signature segment can be used to validate the authenticity of the token so that it can be trusted by your app.
Tokens issued by Azure AD are signed using industry standard asymmetric encryption algorithms, such as RS256. The header of the JWT contains information about the key and encryption method used to sign the token.

```
{
 "typ": "JWT",
 "alg": "RS256",
 "x5t": "iBjL1Rcqzhiy4fpxlxdZqohM2Yk",
 "kid": "iBjL1Rcqzhiy4fpxlxdZqohM2Yk"
}
```

The alg claim indicates the algorithm that was used to sign the token, while the kid claim indicates the particular public key that was used to validate the token.
At any given point in time, Azure AD may sign an id_token using any one of a certain set of public-private key pairs. Azure AD rotates the possible set of keys on a periodic basis, so your app should be written to handle

"kid": "iBjL1Rcqzhiy4fpxlxdZqohM2Yk"
}
The alg claim indicates the algorithm that was used to sign the token, while the kid claim indicates the particular public key that was used to validate the token.
At any given point in time, Azure AD may sign an id_token using any one of a certain set of public-private key pairs. Azure AD rotates the possible set of keys on a periodic basis, so your app should be written to handle those key changes automatically. A reasonable frequency to check for updates to the public keys used by Azure AD is every 24 hours.
You can acquire the signing key data necessary to validate the signature by using the OpenID Connect metadata document located at:
https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration



- configure client credentials using a certificate
- configure Application permissions for the app
- acquire an access token for Microsoft Graph using an application permission and client credential certificate
- acquire an access token using the client secret

Microsoft identity platform allows an application to use its own credentials for authentication, for example, in the OAuth 2.0 Client Credentials Grant flowv2.0 and the On-Behalf-Of flow).
One form of credential that an application can use for authentication is a JSON Web Token(JWT) assertion signed with a certificate that the application owns.

From <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-certificate-credentials>

Build Apps with Microsoft Graph (20-25%)

Optimize Data Usage with query parameters

- use $filter query parameter
- use $select query parameter
- order results using $orderby query parameter
- set page size of results using $skip and $top query parameters
- expand and retrieve resources using $expand query parameter
- retrieve the total count of matching resources using $count query parameter
- search for resources using $search query parameter
- determine the appropriate Microsoft Graph SDK to leverage

Name	Description	Example
$count	Retrieves the total count of matching resources.	/me/messages?$top=2&$count=true
$expand	Retrieves related resources.	/groups?$expand=members
$filter	Filters results (rows).	/users?$filter=startswith(givenName,'J')
$format	Returns the results in the specified media format.	/users?$format=json
$orderby	Orders results.	/users?$orderby=displayName desc
$search	Returns results based on search criteria.	/me/messages?$search=pizza
$select	Filters properties (columns).	/users?$select=givenName,surname
$skip	Indexes into a result set. Also used by some APIs to implement paging and can be used together with $top to manually page results.	/me/messages?$skip=11
$top	Sets the page size of results.	/users?$top=2

Generally speaking, follows Powershell object syntax
The following example shows a query filtered by the **subject** and **importance** properties, and then sorted by the **subject**, **importance**, and **receivedDateTime** properties in descending order.
https://graph.microsoft.com/v1.0/me/messages?$filter=Subject eq 'welcome' and importance eq 'normal'&$orderby=subject,importance,receivedDateTime desc

Optimize network traffic

- monitor for changes using change notifications
- combine multiple requests using $batch
- get changes using a delta query
- implement error 429 handler

Change Notifications
HTTP
POST https://graph.microsoft.com/v1.0/subscriptions
Content-Type: application/json
{
"changeType": "created,updated",
"notificationUrl": "https://webhook.azurewebsites.net/notificationClient",
"resource": "/me/mailfolders('inbox')/messages",
"expirationDateTime": "2016-03-20T11:00:00.0000000Z",
"clientState": "SecretClientState"
}
The changeType, notificationUrl, resource, and expirationDateTime properties are required. See subscription resource type for property definitions and values.
The resource property specifies the resource that will be monitored for changes. For example, you can create a subscription to a specific mail folder: me/mailFolders('inbox')/messages or on behalf of a user given by an administrator consent: users/john.doe@onmicrosoft.com/mailFolders('inbox')/messages.
Although clientState is not required, you must include it to comply with our recommended change notification handling process. Setting this property will allow you to confirm that change notifications you receive originate from the Microsoft Graph service. For this reason, the value of the property should remain secret and known only to your application and the Microsoft Graph service.
If successful, Microsoft Graph returns a 201 Created code and a subscription object in the body.
Note: Any query string parameter included in the **notificationUrl** property will be included in the

Access User data from Microsoft Graph

- get the signed in users profile
- get a list of users in the organization
- get the users profile photo
- get the user object based on the users unique identifier
- get the users manager profile

Access Files with Microsoft Graph

- get the list of files in the signed in users OneDrive
- download a file from the signed in users OneDrive using file unique id
- download a file from a SharePoint Site using the relative path to the file
- get the list of files trending around the signed in user
- upload a large file to OneDrive
- get a user object from an owner list in a group and retrieve that user's files

Although clientState is not required, you must include it to comply with our recommended change notification handling process. Setting this property will allow you to confirm that change notifications you receive originate from the Microsoft Graph service. For this reason, the value of the property should remain secret and known only to your application and the Microsoft Graph service.
If successful, Microsoft Graph returns a 201 Created code and a subscription object in the body.

> **Note:** Any query string parameter included in the **notificationUrl** property will be included in the HTTP POST request when notifications are being delivered.

From <https://docs.microsoft.com/en-us/graph/webhooks>

$batch
Literally just a single JSON POST that contains multiple requests in 1 request. Also allows a nested property of *dependsOn* to make sure that hierarchies are handled if needed.
https://docs.microsoft.com/en-us/graph/json-batching

Delta Query
https://graph.microsoft.com/beta/groups/delta/?$filter=property eq "value"
Returns a *deltaLink* and *nextLink* that can be used to query for only resources that have changed since initial query.

429 Handler
The number of requests performed may negatively impact MS Graph. HTTP Response code 429 "Too Many Requests" is returned with "Retry-After" header containing an integer value for the number of seconds to wait before retrying the request. Retrying before that time period results in a new 429. MS SDK automatically handles these delays responsibly.

Get current users profile: /me
Get users in Organization: /users
Get users Photo (returned in binary): /me/photo/$value
Get users profiles with unique identifier: /users/{id | userPrincipalName}
Get Manager: /contacts/{id}/manager
From <https://docs.microsoft.com/en-us/graph/api/orgcontact-get-manager?view=graph-rest-1.0&tabs=http>

Manage a group lifecycle on Microsoft Graph

- get the information on a group by id
- get the list of members in a Group
- get the list of owners of a Group
- get the list of Groups where the signed in user is a member
- get the list of Groups where the signed in user is an owner
- provision a Group
- provision a Team with a Group
- delete a group

/me/drive

- Drive - Represents a logical container of files, like a document library or a user's OneDrive.
- DriveItem - Represents an item within a drive, like a document, photo, video, or folder.

From <https://docs.microsoft.com/en-us/graph/api/resources/onedrive?view=graph-rest-1.0>

DriveItems are returned as binary. Upload allows items <4MB to be uploaded as single PUT request. Larger than that should *createUploadSession* where sequential chunks can be PUT until file upload is complete.

List trending files: /me/insights/trending
For identifying which financial spreadsheet is trending in your organization for whatever reason.

Groups basically have the same API as users
https://docs.microsoft.com/en-us/graph/api/resources/groups-overview?view=graph-rest-1.0

Deleting a group: DELETE /groups/{id}
Get owners: GET /groups/{id}/owners
Get Membership: POST /groups/{id}/getMemberGroups

Extend and Customize SharePoint (20-25%)

Understand the components of a SharePoint Framework (SPFx) web part

- identify the appropriate tool to create an SPFx Web Part project
- understand properties of client-side web parts
- understand Office UI Fabric in client-side web parts
- understand when to use an app page
- differentiate between app page and web part
- understand rendering framework options

Basically a Single Page App in Sharepoint. It's Client Side, so no storage of secret keys as that can't be done securely. Office UI Fabric reflects O365 Branding, you are recommended to use it. App pages are "full page", web parts are full width of whatever container they're in (think iframes and MS Teams tabs).
Documentation recommends VSCode with npm yeoman generator for templates.

Understand SPFx extensions

- identify the appropriate tool to create an SPFx Extension project
- understand page placeholders from Application Customizer
- understand the ListView Command Set extension
- understand the Field Customizer extension

- **Application Customizers**: Adds scripts to the page, and accesses well-known HTML element placeholders and extends them with custom renderings.
- **Field Customizers**: Provides modified views to data for fields within a list.
- **Command Sets**: Extends the SharePoint command surfaces to add new actions, and provides client-side code that you can use to implement behaviors.

From <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/extensions/overview-extensions>

Understand the process to package and deploy an SPFx solution

- understand the options for preparing a package for deployment
- understand the options for packaging a solution
- understand the requirements of tenant-scoped solution deployment
- understand the requirements of domain isolated web parts
- understand the options to deploy a solution

Super confusing but basically you can deploy to sharepoint online for everyone, publish it to sharepoint for testing and debugging, and scope it to a single tenant, or publish it to AppSource. Hoping this isn't on the test, as best I can tell no one uses this.
https://docs.microsoft.com/en-us/sharepoint/dev/spfx/web-parts/basics/notes-on-solution-packaging

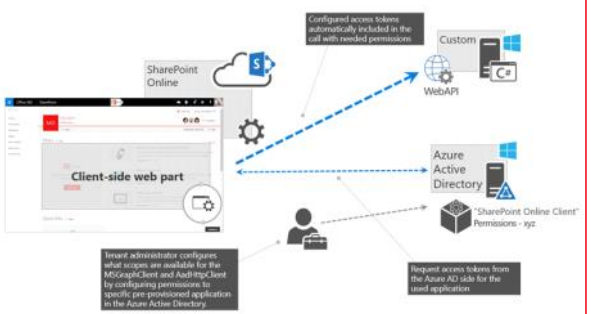
Understand the consumption of Microsoft Graph

- understand the purpose of the MSGraphClient object
- understand the methods for granting permissions to Microsoft Graph

MSGraphClient is a wrapper for a library to ease use of the Graph API (who would've guessed?).
Default is permissions scope is user_impersonation (User.read.All).
Want to access an AzureAD secured API?
Use the **AadHttpClient** with an Oauth flow

From <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/use-aadhttpclient>

Understand the consumption of third party APIs secured with Azure AD from within SPFx



- understand the purpose of the AadHttpClient object
- understand the methods for granting permissions to consume a third party API

Understand Web Parts as Teams Tabs

- understand the considerations for creating a SPFx Web Part to be a Teams Tab
- understand the options for deploying a SPFx Web Part as a Teams Tab

Understand branding and theming in SharePoint

- understand the options for SharePoint site theming
- understand the options for site designs and site scripts

Probably want to update the render() context to not say "welcome to sharepoint App X" when running is MS Teams.
When deploying web parts as Team's tabs, update the manifest.json

- "supportedHosts": ["SharePointWebPart", "TeamsTab"]

Deploy the Sharepoint Web Part
Select the App in the Sharepoint portal and click "Sync to Teams"

Extend Teams (15-20%)

- understand the options for SharePoint site theming
- understand the options for site designs and site scripts

When deploying web parts as Team's tabs, update the manifest.json
- "supportedHosts": ["SharePointWebPart", "TeamsTab"]

Deploy the Sharepoint Web Part
Select the App in the Sharepoint portal and click "Sync to Teams"

Extend Teams (15-20%)

Understand the components of a Teams app

- understand the purpose of a Teams app manifest
- understand App Studio functionality and features
- identify the components of an app package for Microsoft Teams
- understand the options for distributing a Teams app
- understand the benefits of using deep links
- understand task modules

You can use any styling framework you want, but Office Fabric UI is heavily "recommended" to reflect Office branding. Therefore, it's going to be the correct answer. Other than that, basically make sure styling defaults to 100% max-width and 320px min-width of container to support re-flow. Also they "recommend" using theme styling so that your SPFx app inherits the colors used in whatever site it is deployed on.

Teams App manifest lists properties of the app (name, versions, etc…) as well as references the app icons location.
App studio allows you to quickly create and app: Supports editing manifest, Card editor, Control Library
Manifest determines whether it's a Team Tab, Personal tab, or both
Allows adding a conversation bot you've already created with "Bot Framework"
Deep links are massive URL's that can point a user to any nested resource with MS Teams

Understand webhooks in Microsoft Teams

- understand when to use webhooks
- understand the limitations of webhooks
- understand the differences between incoming and outgoing webhooks

Webhooks are an easy way to tie in with external applications. Outgoing webhooks are essentially a POST request that can be triggered from withing teams. A Card can have a button "Approve Invoice" that automatically POSTs an API that approves the invoice.
Incoming webhooks are a MS provisions endpoint that receive JSON POST data and automatically generate a message withing Teams for other users to read.

Understand tabs in Microsoft Teams

- understand when to use tabs
- understand the capabilities of personal tabs
- understand the capabilities of channel tabs
- understand the requirements for tabs for mobile clients

Tabs should be used to access external data not available in Teams. Personal tabs should be used when the data concerns one specific user (individual time tracking) and Channel tabs should be used when the data concerns multiple users within the channel (changes to a meeting time).
Tabs for mobile must be installed on desktop or web clients and can take 24 hours to show up on mobile. MS has a lot to say about styling (list, grid, etc..) for mobile tabs, but aside from that really the only other thing you need to be concerned with is bandwidth/data usage for the tab.
https://docs.microsoft.com/en-us/microsoftteams/platform/tabs/design/tabs-mobile

Understand messaging extensions

- understand when to use messaging extensions
- understand where messaging extensions can be invoked from
- understand search based messaging extensions
- choose the appropriate message extension command type based on requirements

Messaging extensions can be called from within the search bar or from within a Teams Message Composition window. You can have certain commands like /news post a news story card, or have a search based messaging extension look up a client's contact information. For /news, that extension probably belongs in a Teams channel. For /contactinfo it allows you to quickly query an external data source with a parameter for information to post as message.

Adaptive Cards can implement webhooks, images, and text blurbs to allow other users to quickly perform actions on the newly provided information.

- understand action-based messaging extensions with adaptive cards
- understand action-based messaging extensions with parameters

Understand conversational bots

- understand when to use conversational bots
- understand the scoping options for bots
- understand when to use a task module from a bot

Bots can be tagged with @botName and trigger a conversational bot.
When used in a Teams channel, they should only be used with there is a 1:1 request/response cycle where the information is relevant to multiple users. Examples of this include notifications, polls, surveys.
More complex workloads belong in a one-to-one chat with the bot.
Task modules allow you to create modal popup experiences in your Teams application. Inside the popup you can run your own custom HTML/JavaScript code allowing for dynamic user input, show an <iframe>-based widget such as a YouTube or Microsoft Stream video or display an Adaptive card.

From <https://docs.microsoft.com/en-us/microsoftteams/platform/task-modules-and-cards/what-are-task-modules>

Extend Office (15-20%)

Understand fundamental components and types of Office Add-ins

- understand task pane and Content Office Add-ins
- understand dialogs
- understand custom functions
- understand Add-in commands
- understand the purpose of Office Add-ins manifest

Task panes are interface surfaces that typically appear on the right side of the window within Word, PowerPoint, Excel, and Outlook.
Dialog is basically an HTML pop-up. Used for authentication, etc…
Custom Functions are literally that.
Commands are nice UI buttons you get in the Office UI ribbon
Manifest defines icons, name, permissions, version, id…

Understand Office JS APIs

- understand the Office Add-in programming model
- understand Office Add-in developer tools
- understand the capabilities of the Excel Javascript API
- understand the capabilities of the Outlook JavaScript API
- understand the capabilities of the Word JavaScript API
- understand the capabilities of the PowerPoint JavaScript API
- understand the capabilities of custom functions

There's a lot here. Just read this: https://docs.microsoft.com/en-us/office/dev/add-ins/reference/javascript-api-for-office

JS API's for each Office app start with Excel., Word., Powerpoint., etc…
There is also an Office. Common API reference.

There's no way these are actually on the test at any real depth. This is probably good enough.

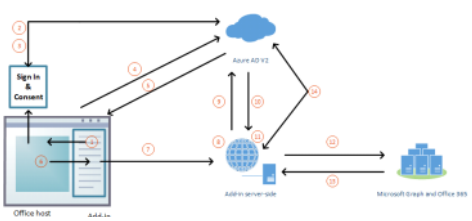
Understand customization of Add-ins

- understand the options of persisting state and settings
- understand Office UI Fabric in Office Add-ins
- understand when to use Microsoft Graph in Office Add-ins
- understand authorization when using Microsoft Graph in Office Add-ins

Understand testing, debugging, and deployment options

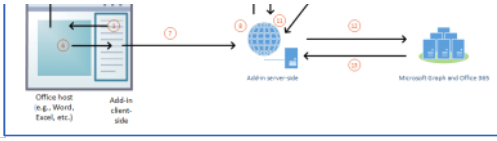
- select deployment options based on requirements
- understand testing and debugging concepts for Office Add-ins

Persist state using the Storage API, or if using Outlook *customProperties*
Yes, yes, *Office Fabric UI reflects Office Branding.* The fact that you can use custom styling isn't gonna be on the test as a valid answer, they'll be asking about Office UI Fabric.
Microsoft Graph allows users to pull data from secured API's



Understand actionable messages

Understand actionable messages



Develop add-in with one of the 2 following options:

- Use the Yeoman generator for Microsoft Office Add-ins to create the project, then open the project in Visual Studio Code
- From a Microsoft Visual Studio that has the Microsoft Office/Sharepoint development workload installed, create a new project that uses Visual Studio Tools for Office (VSTO) template

Custom functions in add-ins can only reliably be debugged from the visual Studio Code implementation.

- understand the features of actionable messages with an adaptive card
- understand the scenarios for refreshing an actionable message

Actionable messages with an adaptive card allow a user to perform an action from within the Office application. Such as receiving an email about an invoice to be paid, and an actionable message using the Action.Http method on the adaptive card to automatically approve it without the user needing to leave the application.

Action.Http is also the preferred way that can be auto-invoked when opening a message that can refresh content of the adaptive card with stuff like "has this invoice already been approved" or "whats the current time in Japan from which our accounting department has sent this"