Contents

There are many nuances to making an offline application work. The DevForce cache is a good start, but it only gets you half way there (or less).

That being said, we've written our fair share of offline-capable applications and we hope that you'll take the following considerations into account before building your own offline-capable apps.

Plan for disconnected states

The application must gracefully fallback if its connection is lost. If a query or save fails against the server, the changes should be persisted locally so they don't get lost. The UI should fall back to only things that can be performed while offline.

Reliable detection of a lost connection is required and the UI must be notified so it can adjust appropriately. Keep in mind that you may only find out that the connection was lost the next time a query or save is attempted. You'll need to think about how the query or save should recover from this.

Manage conflict resolution

Periodic reconnect attempts should be made centrally and after a new connection has been successfully established, any offline changes need to be synced. There can be conflicts at this point. What is your plan to resolve them?

Increase storage

Increasing the isolated storage quota can only happen as part of a UI event (user clicking on a button). If the application can't save the cache in the background, the user will needs to first initiate the increase of the quota before saving can be attempted again.

Your app can ask for ISO capacity upgrade if it has elevated permissions. Otherwise, it may be wise to require the user to push the **Raise capacity** button upon installation of the application. Don't allow any work until the user increases capacity to the acceptable level.

Offline authentication

At a minimum, previously logged in users should be able to login even if they have no connection to the server. How will you securely store their credentials to achieve this? After the connection comes back, they need to be transparently authenticated against the server.

Here's one way to do it:

1. Take the complete Principal and LoginCredentials and stuff it into a serializable object. Call this the offline token.
2. Serialize the token and encrypt it using a hash of the users password.
3. Store the encrypted token in isolated storage. You can use the username as the filename, so that multiple application users can share the same windows account.

Perform the steps above after a successful *online* login. Then, if the user logs in *offline* do the following:

- Read the encrypted token back from isolated storage. The username will be the file name.
- Attempt to decrypt the content using the hash of the user's password. If decryption fails, login fails.
- Deserialze the decrypted token in order to get the principal and credentials back.

To make this application agnostic, you can hide all this behind a security service and never obtain the Principal directly from an EntityManager. The Principal property on the EntityManager will be null until you actually authenticate the user against the server. You should obtain the Principal from this security service. It will either return you the offline Principal it got from the file or return you the actual Principal form an EntityManager.

Then when the application comes online, don't forget to login against the server and propagate the security context to all EntityManagers before attempting a request to the server.

Encrypt the cache file

You should encrypt the cached file as you write it. You don't want a stranger pulling that data from the machine and you don't want to store the encryption key on the client either. See Secure offline data

Version changes to the model

The cache can no longer be deserialzed if the application gets upgraded and there were material changes to the domain model.

This is also true if the DevForce version is upgraded, as we make no guarantee that the internal format of the EntityCacheState will remain unchanged.

Try to use offline storage for limited time only, or give users plenty of notice before a breaking change.

You can also write code to load both model versions and perform a migration.