Story UFC-283: Data Backup/Restore

The following document will go over the design of user account data backup and restore, as outlined in Story ticket UFC-283.

Acceptance Criteria

For reference, the following are the acceptance criteria for this ticket:

- I want to be able to export all of my data as a single plain JSON file for use with other applications.
- I want to be able to export an encrypted backup of my data.
- I want to be able to restore a backup of my data that was made in plain JSON.
- I want to be able to restore an encrypted backup of my data.
- I want my backups to have a schema/version number, so that any future changes are known and backwards compatible for restoration purposes.

Design Brainstorming

So how are we gonna create these backups? More precisely, what are these backups gonna consist of?

What's in the Backups

Well, I think the easy starting point is backup all of the accounts, transactions, import profiles, and import profile mappings. That's an easy decision.

The harder decision is whether or not we backup any of the user model data -- more specifically, email, EDEK, and KEK Salt.

re: email, I don't think there's much merit in backing it up; it's not tied to any of the data (in the way the encryption keys are) and it wouldn't really make sense to 'restore' an old email address if the user has changed the address for their account.

re: EDEK and KEK Salt, I definitely think there's some merit to backing them up, just in the spirit of user data ownership. However, I think there's an argument to be made that there's not really any point in backing them up when the user performs an unencrypted backup -- they don't *need* the keys, since they have the data right there.

However, since they *can* backup encrypted versions of their data, I guess that's where the main merit in allowing them to backup the keys are -- so that they can decrypt their own

encrypted backups. Does that really make sense, since they'll just *have* the unencrypted data? Not really, no.

So what if we look at this from a security perspective. Would it be bad if a user's unencrypted backup (that contained their EDEK/KEK Salt) got leaked? Yes, but the fact that the keys were in there doesn't *really* matter, cause their data would have already been leaked as well!

OK, now what if we look at this from the perspective of a pure data recovery option. Say we lost *all* their data -- their whole user account. Or even say, what if they deleted their user account but then wanted to come back. What about then?

- Well, if they have an unencrypted backup, then they won't need the keys since the data will just be encrypted with whatever their new account's keys are.
- If they have an encrypted backup, then they're screwed either way -- if the keys are in the backup, well we can't read it since it's encrypted, and if they aren't, well they're still screwed.
- But what if they have both? Well then it *doesn't matter*, because they'll just use the unencrypted version.

So then yeah, I just don't think it matters whether or not we give them in unencrypted backups. If anything, it probably makes more sense to offer just a separate option to download *just* the keys, so that they can have the keys and encrypted backups but not unencrypted backups. This way, if worst comes to worst, they can always decrypt their own data (which would be really convoluted if they wanted to do this outside of uFincs, since the EDEK/KEK Salt aren't *actually* keys, and the decryption process isn't trivial, but those are minor details).

So, I guess for this ticket, we'll just offer unencrypted and encrypted backups, with no keys, and add the caveat to encrypted backups that they can only be restored to this user account (whereas unencrypted backups can be restored to any account).

I think that makes sense. If users want the keys, then I'm sure they'll let us know.

If I ask myself "what would I want", then I'd probably be fine with *just* the unencrypted backups, since I trust myself to take care of my data, I don't actually care if this particular data is leaked, and unencrypted backups can easily be restored to any uFincs account. So there.

How to Make Backups

This should be fairly straightforward. To get the data for the backups, we just need a single cross slice selector that selects across all of the slices that we want to backup. We'll basically be doing a dump of the relevant slices into a single JSON blob as the 'backup schema'. Of course, we can't forget to version it, in case this somehow bites us in the back in the future and we want to change it.

So the final file format might be as simple as:

```
{
    version: "1.0",
    data: {...store dump...}
}
```

Then, in terms of making an encrypted backup, we just need to take the store dump and run each slice through the encryption middleware. We could then go one step further and combine the whole thing into one base64 encoded string (if we wanted to obfuscate the store format), but that's probably overkill? I mean, the store format is already effectively public (this is the client side, after all), so we probably don't need to do this. Might as well keep it simple and just dump it straight from the encryption middlware (such that only the fields are encrypted).

I just had a thought: we need to be able to differentiate the the encrypted vs the not encrypted file types. First thought was to add something like a `encrypted` attribute to the top-level JSON blob, but we could also just modify the version tag to support it. Something like `1.0-encrypted` and `1.0-unencrypted`. Nah, let's just do an attribute. Then we can assume 'unencrypted' if the attribute was somehow removed.

So now the schema looks like:

```
{
    version: "1.0",
    encrypted: true/false,
    data: {...store dump...}
}
```

How to Download Files

OK, apparently creating and downloading a file purely on the frontend isn't as trivial as I thought it was going to be. I mean, it's *easy* to do, it's just like "really, there isn't a better way to do this?".

Basically, you create a Blob object as the 'file' and attach it as the `href` on an anchor tag, where the anchor tag has the `download` attribute set. See https://stackoverflow.com/questions/44656610/download-a-string-as-txt-file-in-react/44661948, Create json file using blob.

Yeah, that seems kinda hacky. And also something that can't really be handled directly in a saga, since we kinda don't have DOM access in a saga. So... I guess we'll have some sort of dedicated 'file download' slice, that then maps to some hidden anchor tag so that we can trigger downloads just by writing to the slice state? Yeah, that makes sense.

How to Restore Backups

In either case, restoring a backup means wiping all existing data. That's easy enough from a frontend perspective (just calling the `set` action for each slice is enough), but slightly less easy for the backend.

We almost need some way to just invoke the `deleteUserData` hook. My first thought would be to have a dedicated service just for this, but that seems excessively overkill.

Perhaps we could just have send a custom `query` param for the user service `remove` method, so that it performs the user data deletion but then bails out before actually removing their account (i.e. by setting `context.result` -- see Hooks | FeathersJS)? That could work.

Then we'd need to be able to re-create the data. This is almost like the encryption migration process, except we don't need to fetch the user data from the backend first -- just query it up from the store, issue an action to encrypt it, and then send the encrypted data to the backend.

In terms of how to decrypt an encrypted backup, that should be so trivial as to not have to write down, but I will for posterity's sake -- just take each slice from the backup and, when performing the `set` actions, make sure they have the `decrypt` meta tag.

Which saga class handles all this stuff?

Like, should data backup/restores have its own dedicated slice + sagas? Not really, unless we want to dedicate the file download slice as the backup slice.

I think the backup stuff should fall under the `user` sagas.

Actually, now that I think about it, the shouldn't the change email/change password/delete user account sagas *also* fall under the `user` sagas? Instead of the `auth` sagas? Hmm... Yeah, they probably should.

Big Problem...

So, I've only just now realized that restoring a backup to another user account is quite problematic... Why? Because the IDs are globally unique! I can't just re-create the accounts and transactions on another user account if their IDs are all the same as another user's! FUCK.

I mean, I can always just rewrite the IDs when restoring a backup, but I don't want to...

Update: Ended up just rewriting the IDs anyways. Always. Why? Because that's the easiest thing from my perspective even though it hurts potential integrate-ability for end-users (since their IDs would always change whenever they backup/restore).

Component Breakdown

Atoms

- N/A

Molecules

- N/A

Organisms

- BackupDataForm
- RestoreDataForm

Scenes

- Settings → MyData

Tasks

- Build the layout.
- Make it work.
- Remove the "CSV Import Formats" section, since it's not going to be implemented yet, and we don't exactly need a full ticket to do this.