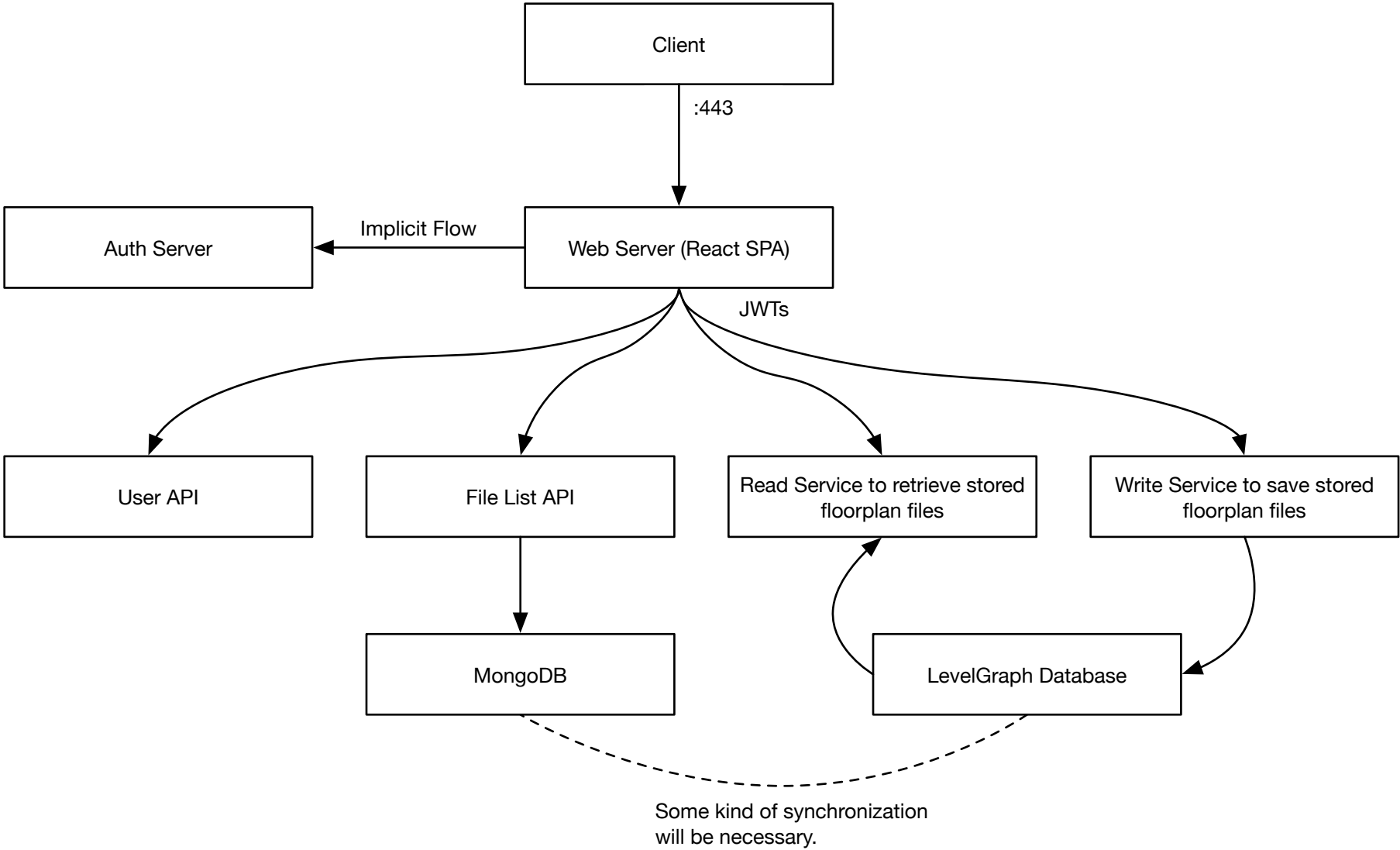
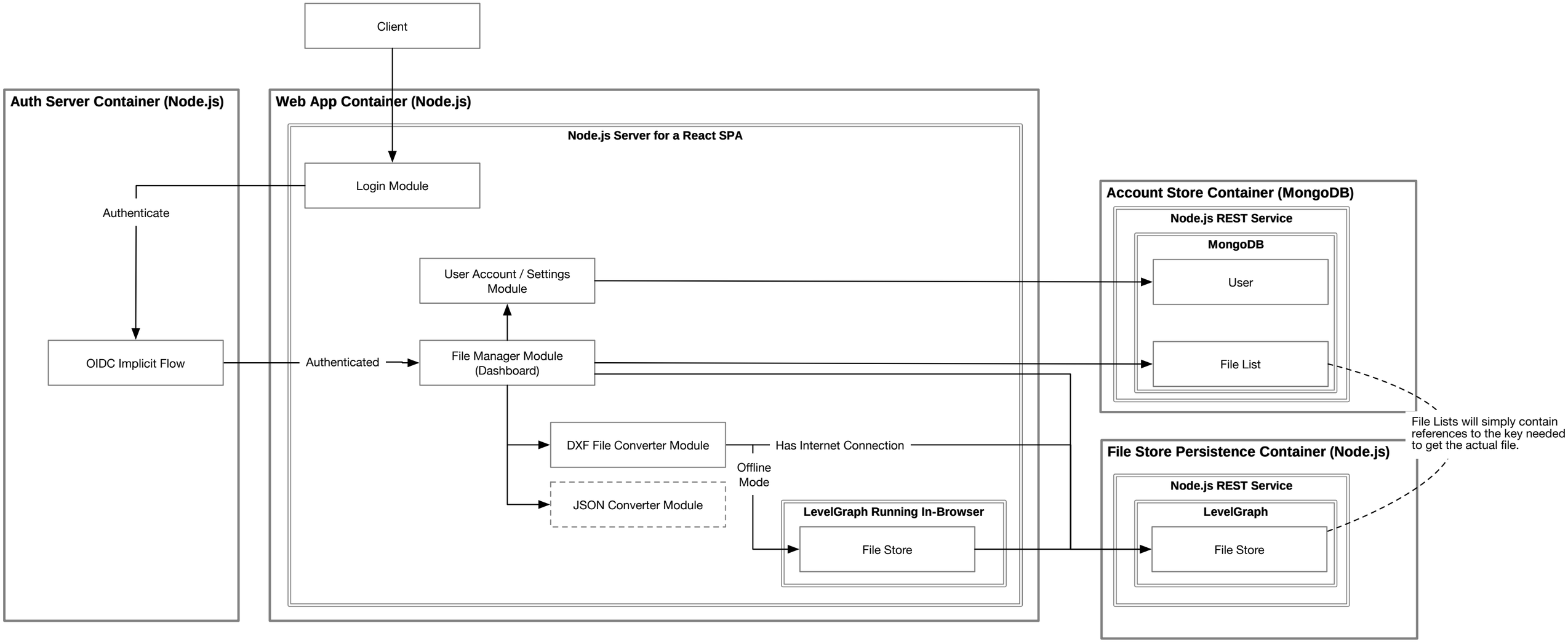
System Diagram



Application Diagram



Authentication / Authorization
- Ideally this would be a simple cloud-based solution like Okta's very nice OIDC implementation.
- Alternatively we may want to use something like the `openid-client`.
- All that this service needs to do is authenticate the user and help decide what kind of access the user has to modify files.
- This will probably be more complex than simple read / write because we'd like to allow offline. Maybe that solution will simply to be forcing the user to clone a file to modify it if they have not been granted more than read access.

Login Module
- A thin UI that redirects to a third-party OIDC server for actual authentication.

User Account / Settings Module
- For setting user preferences, user name, avatar, etc.

File Manager Module
- List, rename, clone, delete existing files

DXF File Converter
- UI that handles the upload, parsing, and conversion of DXF files to simpler JSON format.
- The idea is that this can all be run in the browser via JavaScript.
- Using a tool like LevelDB will allow us to interact with the data store directly in the browser as well.
- We can leverage Level's graph version to better fit the complex relationships between the system elements.

JSON Converter
- Convert JSON back to DXF
- This module may not be necessary

Persistence
- Looking at LevelGraph for its ability to run in-browser.
- It's a graph database built on top of LevelDB, which is a fast NoSQL key-value store.
- Graph databases will help manage the complex relationships between DXF components.
- Offline capabilities are great because of possible limitations for where the technicians working with these files may be operating (e.g. behind a firewall).

File Store Persistence
- The idea here is that if and when the user gets internet access back, the app will back up the files.
- This will probably be sitting behind a REST service

API Planning

Notes
- Hashes for IDs
- Need to define the JSON format
- Streams to serve up file