const subscribableTree = SubscribableTreeLoader.load(treeId) TREE_LOADER const firebaseTreesRef = firebase.database().ref('trees/') // rather this would be injected + load(treeld) from the inversify.config object graph const treeRef = firebaseTreesRef.child(treeId) const contentIdRef = firebaseTreesRef.child('contentId') load(treeld) implementation: const parentIdRef = firebaseTreesRef.child('parentId') const json = await getTreeJson(treeId) const childrenRef = firebaseTreesRef.child('children') const deserializer = new SubscribableTreeDeserializer(treeId, json) //TODO: I wish I const contentIdSaver = new FirebaseSaver({contentIdRef}) //TODO: avoid new. Somehow could deserialize the normal tree, and then apply the can we inject this or use a factory or something? subscribable behaviors onto the normal tree to transform it const parentIdSaver = new FirebaseSaver({parentIdRef}) //TODO: avoid new. Somehow can into a subscribable tree we inject this or use a factory or something? const tree = deserializer.deserialize() const childrenSaver = new FirebaseSaver({childrenRef}) //TODO: avoid new. Somehow can return tree we inject this or use a factory or something? const contentIdSyncerToDB = new SyncToDB({saveUpdatesToDBFunction: //now contentIdSaver.save}) const parentIdSyncerToDB = new SyncToDB({saveUpdatesToDBFunction: parentIdSaver.save}) const childrenSyncerToDB = new SyncToDB({saveUpdatesToDBFunction: childrenSaver.save}) SUBSCRIBABLE_TREE_DESERIALIZER (json) + constructor(json) + deserialize() : ISubscribableBasicTree const dbSubscriberToTree = new DBSubscriberToTree(subscribableTree, contentIdSyncerToDB, parentIdSyncerToDB, childrenSyncerToDB) dbSubscriberToTree.subscribe() const dbSubscriberFromTree = new DBSubscriberFromTree(subscribableTree,

-transforms those updates and publishes them to any subscribers
-has an onUpdate method to register subscribers
>

-subscribes to updates from firebase ref

SubscribeToFirebase class

Route '/', component: BranchesAppComponent BranchesAppComponent.ts async created() { this.init() const app = myContainer.get<IBranchesApp>(TYPES.IBranchesApp) // will inject the correct databases, localStorage, localStorageRetrievers etc. app.setURL(getWindowURI()) app.init() BranchesApp.ts this.datastore.init() // DataStore has LocalStorageHandler already injected into it this.ui_bridges.subscribe(this.datastore) this.getContentItem UIBridges.ts TreeUIBridge ContentItemUIBridge subscribe(subscribable: Isubscribable) subscribable.onUpdate(this.handleDataUpdate) handleDataUpdate({dataUpdate: IDataUpdate}) { switch (dataUpdate.datatype) { case DATA_TYPES.TREES: treeUIBridge.handleDataUpdate({dataUpdate}) case DATA_TYPES.TREE_USER_DATA treeUserDataUIBridge.handleDataUpdate({dataUpdate}) TreeUIBridge.ts { handleDataUpdate({dataUpdate: IDataUpdate}) { const sigmaNodeId = DATA_SIGMA_MAP.getFromTreeId(dataUpdate.treeId) sigmaHandler.update({sigmaNodeId: dataUpdate.treeId, updateType: DATA_TYPES.TREES, newTreeData: dataUpdate.val}) TreeUserDataUIBridge.ts { handleDataUpdate({dataUpdate: IDataUpdate}) { const sigmaNodeId = DATA_SIGMA_MAP.getFromTreeId(dataUpdate.treeId) sigmaHandler.update({sigmaNodeId: dataUpdate.treeId, updateType: DATA_TYPES.TREE_USER_DATA, newTreeUSerData: dataUpdate.val}) ContentItemUserDataUIBridge.ts { handleDataUpdate({dataUpdate: IDataUpdate}) { const sigmaNodeIds = DATA_SIGMA_MAP.getFromContentId(dataUpdate.contentId) sigmaNodeIds.forEach(sigmaNodeId => { sigmaHandler.update({sigmaNodeId: dataUpdate.treeId, updateType: DATA_TYPES.TREE_USER_DATA, newTreeUSerData: dataUpdate.val})

DataStore.ts() {

ACTIONS

